(12) UK Patent Application (19) GB (11) 2 387 754 (13) A

(43) Date of A Publication 22.10.2003

(21) Application No 0208727.8

(22) Date of Filing 17.04.2002

(51) INT CL[7]
G06T 15/10

(52) UK CL (Edition V)
H4T TBBA T143

(56) Documents Cited
WO 2000/073888 A1 US 4685070 A
JP 2001204922 A

(58) Field of Search
Other: ONLINE: WPI EPODOC PAJ

(71) Applicant(s)
John Morrison
28 Cambria Place, SWINDON, Wiltshire,
SN1 5DN, United Kingdom

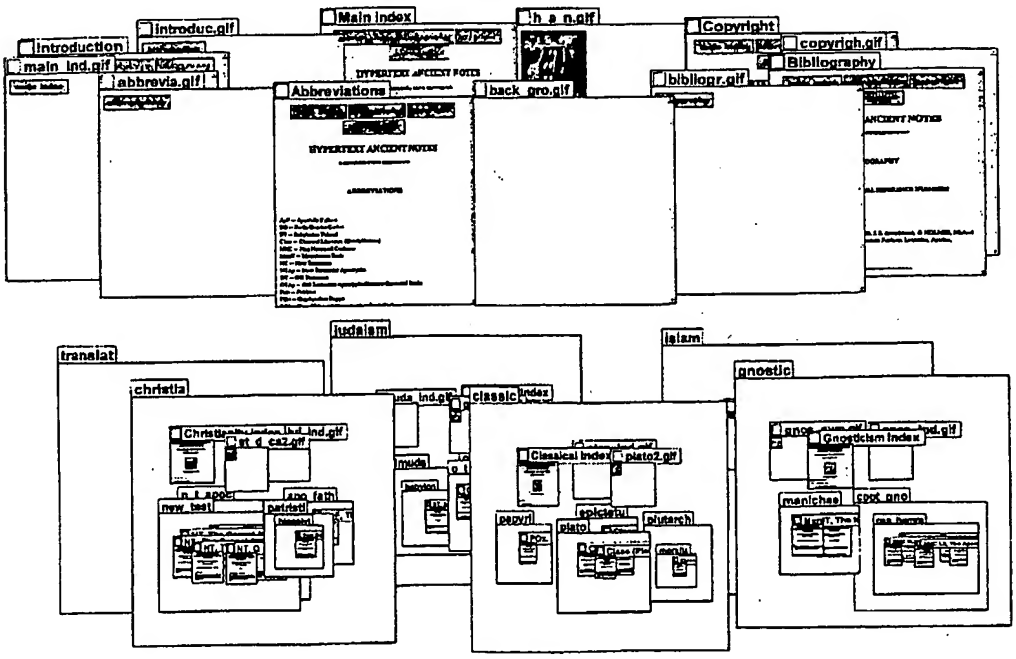
(72) Inventor(s)
John Morrison

(74) Agent and/or Address for Service
Marks & Clerk
4220 Nash Court,
Oxford Business Park South, OXFORD,
OX4 2RU, United Kingdom

(54) Abstract Title
Representing information as 3D image

(57) Digitally stored information is displayed by creating a virtual 3D space which is rotatable in response to user interaction. Items of information are represented as labelled nodes within the 3D space which appear attached to their respective nodes. The user receives the impression of a set of 2D images suspended within the 3D space, so that the 2D images always face the user whatever the orientation of the 3D space. Information arranged as a hierarchical tree of files and folders can be displayed using a root 3D space to display images of the files and folders listed in the root of the tree, with further folders and associated files displayed using a further 3D space. Links between the files can be shown as associative lines (see fig. 10).

Fig. 3



BEST AVAILABLE COPY

GB 2 387 754 A

Fig. 1

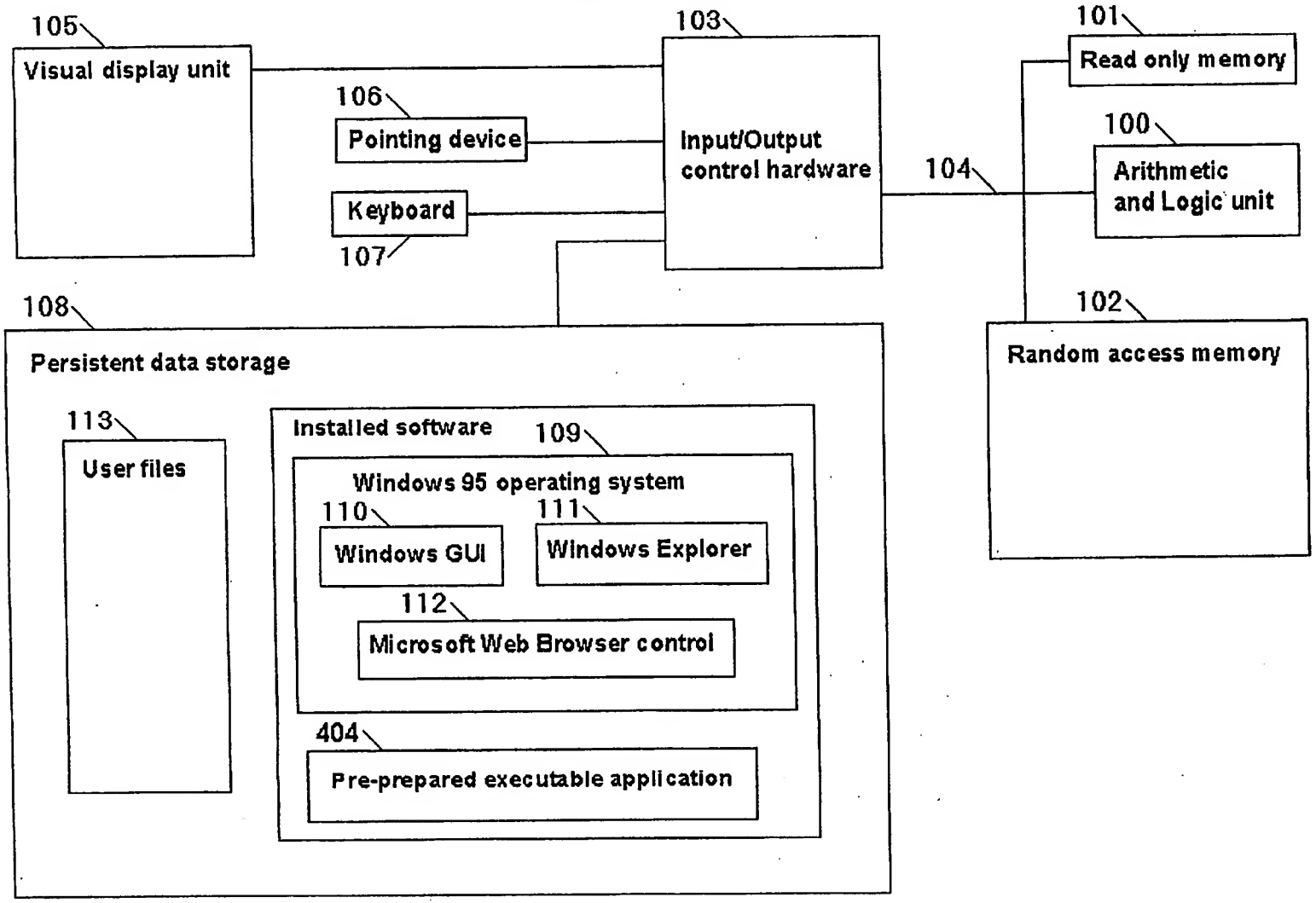

105
Visual display unit
106
Pointing device
Keyboard
107
103
Input/Output control hardware
104
101
Read only memory
100
Arithmetic and Logic unit
102
Random access memory
108
Persistent data storage
113
User files
Installed software
109
Windows 95 operating system
110
Windows GUI
111
Windows Explorer
112
Microsoft Web Browser control
404
Pre-prepared executable application

Fig. 2

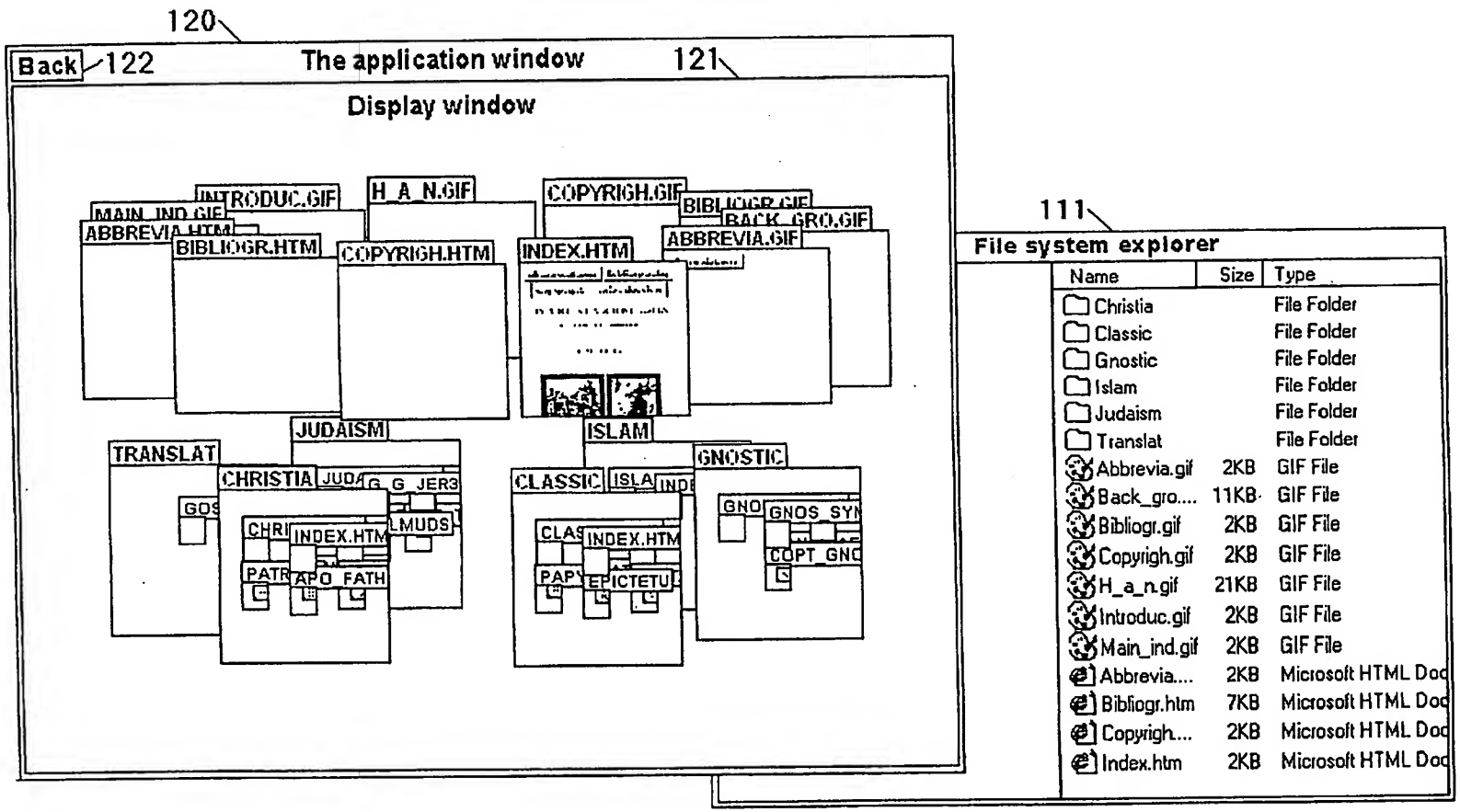
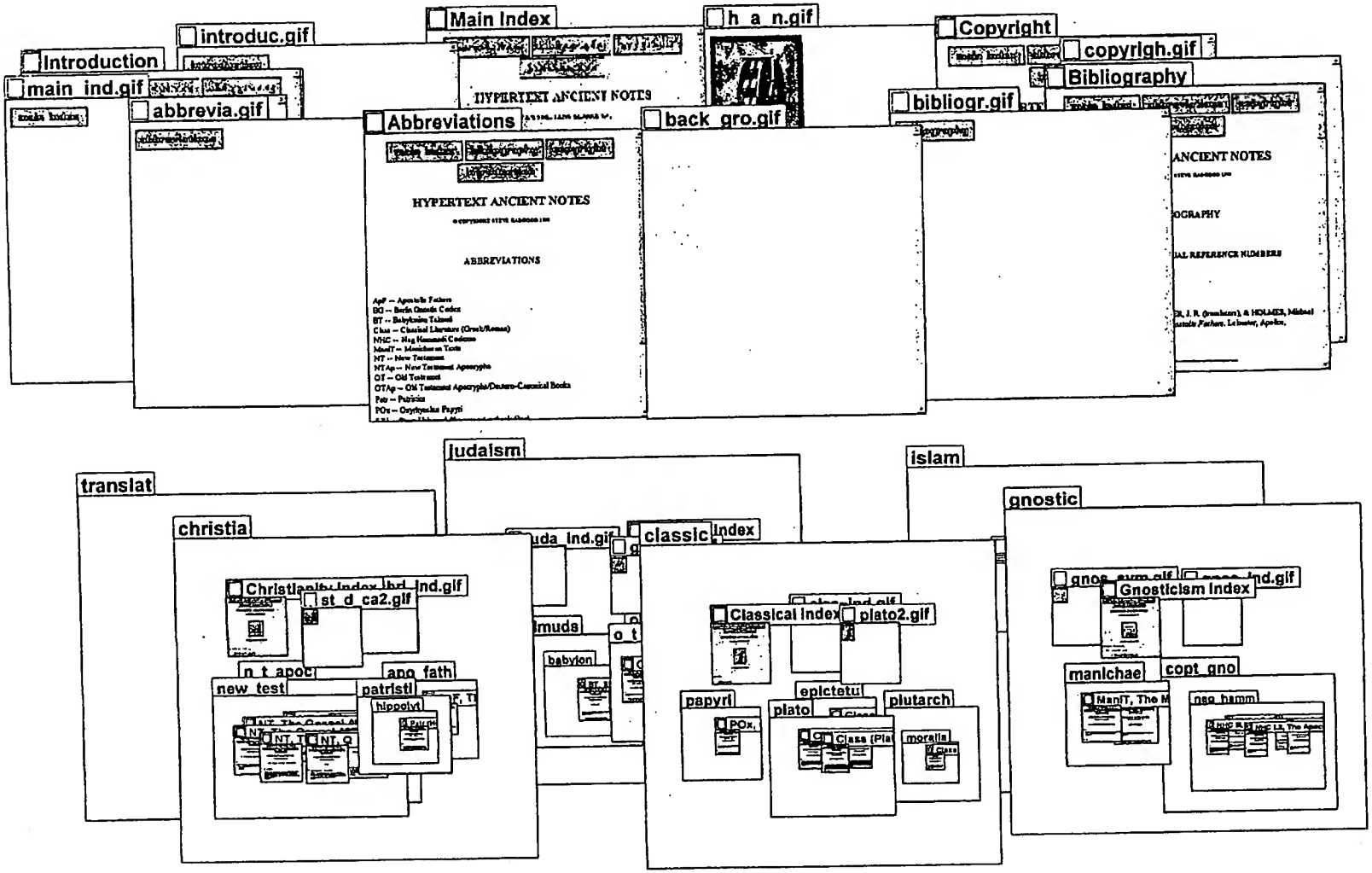

120
Back
122
The application window
121
Display window
INTRODUC.GIF
H_A_N.GIF
COPYRIGH.GIF
BIBLIOGR.GIF
BACK_GRO.GIF
MAIN_IND.GIF
ABBREVIA.HTM
BIBLIOGR.HTM
COPYRIGH.HTM
INDEX.HTM
ABBREVIA.GIF
JUDAISM
ISLAM
TRANSLAT
CHRISTIA
CLASSIC
GNOSTIC
INDEX.HTM
111
File system explorer
Name
Size
Type
Christia
File Folder
Classic
File Folder
Gnostic
File Folder
Islam
File Folder
Judaism
File Folder
Translat
File Folder
Abbrevia.gif
2KB
GIF File
Back_gro....
11KB
GIF File
Bibliogr.gif
2KB
GIF File
Copyrigh.gif
2KB
GIF File
H_a_n.gif
21KB
GIF File
Introduc.gif
2KB
GIF File
Main_ind.gif
2KB
GIF File
Abbrevia....
2KB
Microsoft HTML Doc
Bibliogr.htm
7KB
Microsoft HTML Doc
Copyrigh....
2KB
Microsoft HTML Doc
Index.htm
2KB
Microsoft HTML Doc

Fig. 3


Introduction
main_ind.gif
introduc.gif
abbrevia.gif
Main Index
HYPERTEXT ANCIENT NOTES
h_a_n.gif
Abbreviations
HYPERTEXT ANCIENT NOTES
ABBREVIATIONS
back_gro.gif
Copyright
copyrigh.gif
Bibliography
bibliogr.gif
ANCIENT NOTES
translat
christia
judaism
islam
gnostic
classic
index
Classical Index
plato2.gif
papyri
plato
epictetu
plutarch
moralia
Gnosticism Index
manichae
copt_gno
new_test
patristi
hippolyt
babylon

3/26

Fig. 4

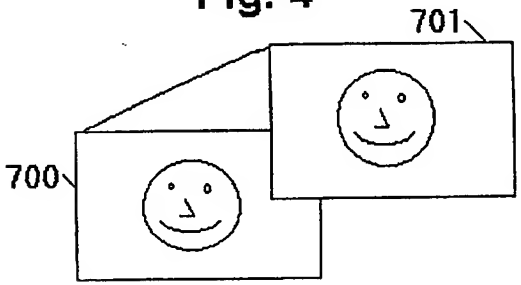


Fig. 5

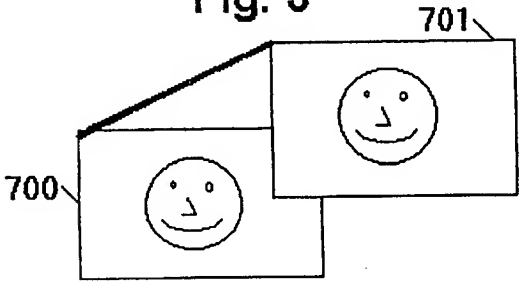


Fig. 6

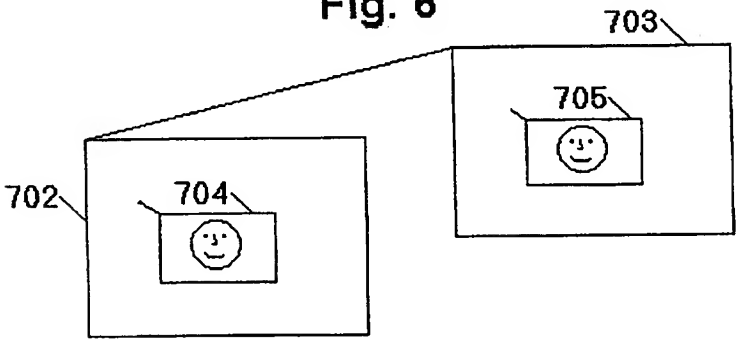


Fig. 7

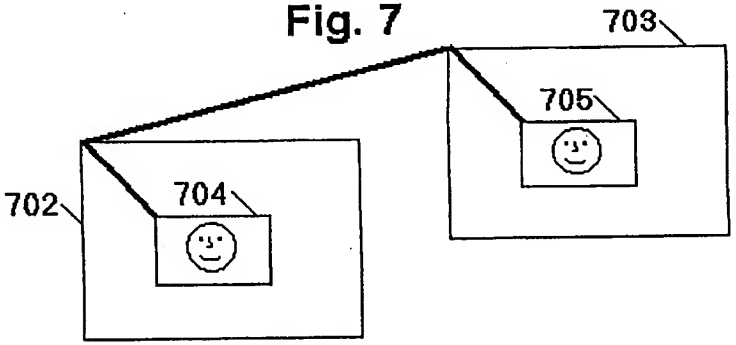


Fig. 8

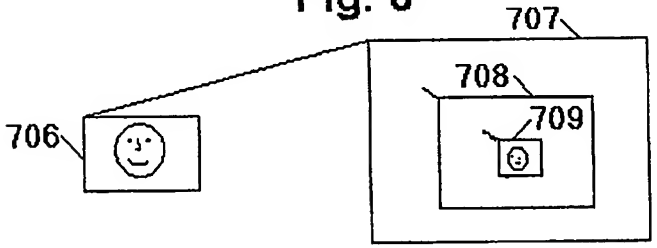
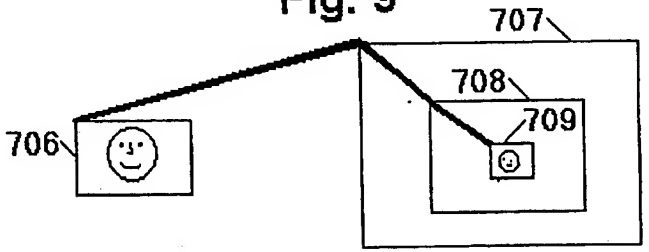


Fig. 9



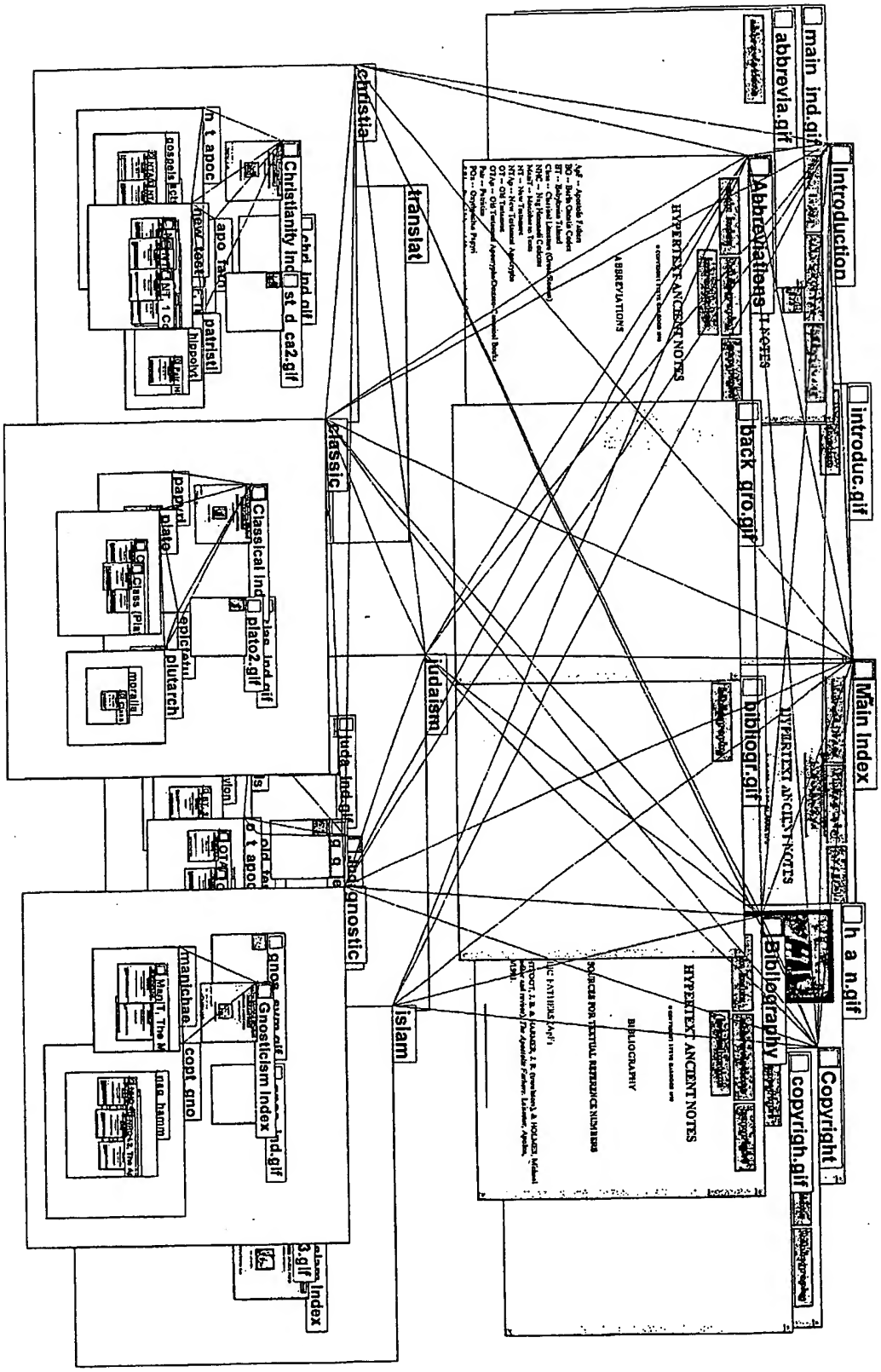

main_ind.gif
abbrevia.gif
Introduction
Abbreviations
introduc.gif
back_gro.gif
Main Index
bibliogr.gif
h a n.gif
Bibliography
Copyright
copyrigh.gif
christia
translat
classic
judaism
gnostic
islam
Christianity Ind
chri_ind.gif
st_d
ca2.gif
Classical Ind
plato2.gif
Gnosticism Index
3.gif
Index

Fig. 10

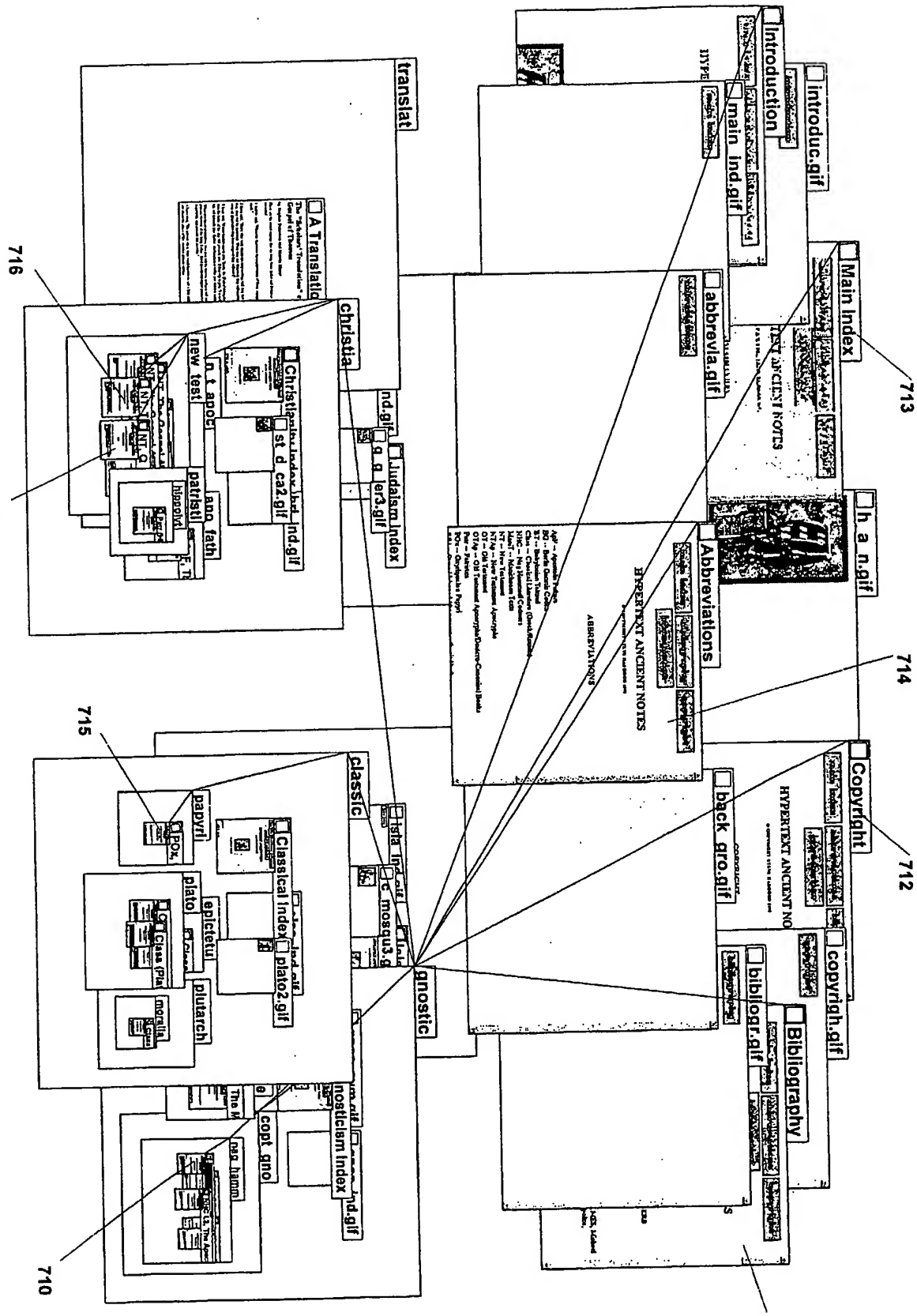

Introduction
introduc.gif
main_ind.gif
Main Index
abbrevia.gif
h a n.gif
Abbreviations
Copyright
copyright.gif
back_gro.gif
Bibliography
bibliogr.gif
translat
A Translatio
christia
new test
n_t_apoc
st_d_ca2.gif
patristi
lang_fath
Hippolyt
nd.gif
Judaism Index
classic
papyri
POx
Classical Index
plato
epictetu
plato2.gif
plutarch
gnostic
c_mosqu3.g
gnosticism Index
copt_gno
nag_hamm
713
714
712
711
716
717
715
710

Fig. 11

Fig. 12

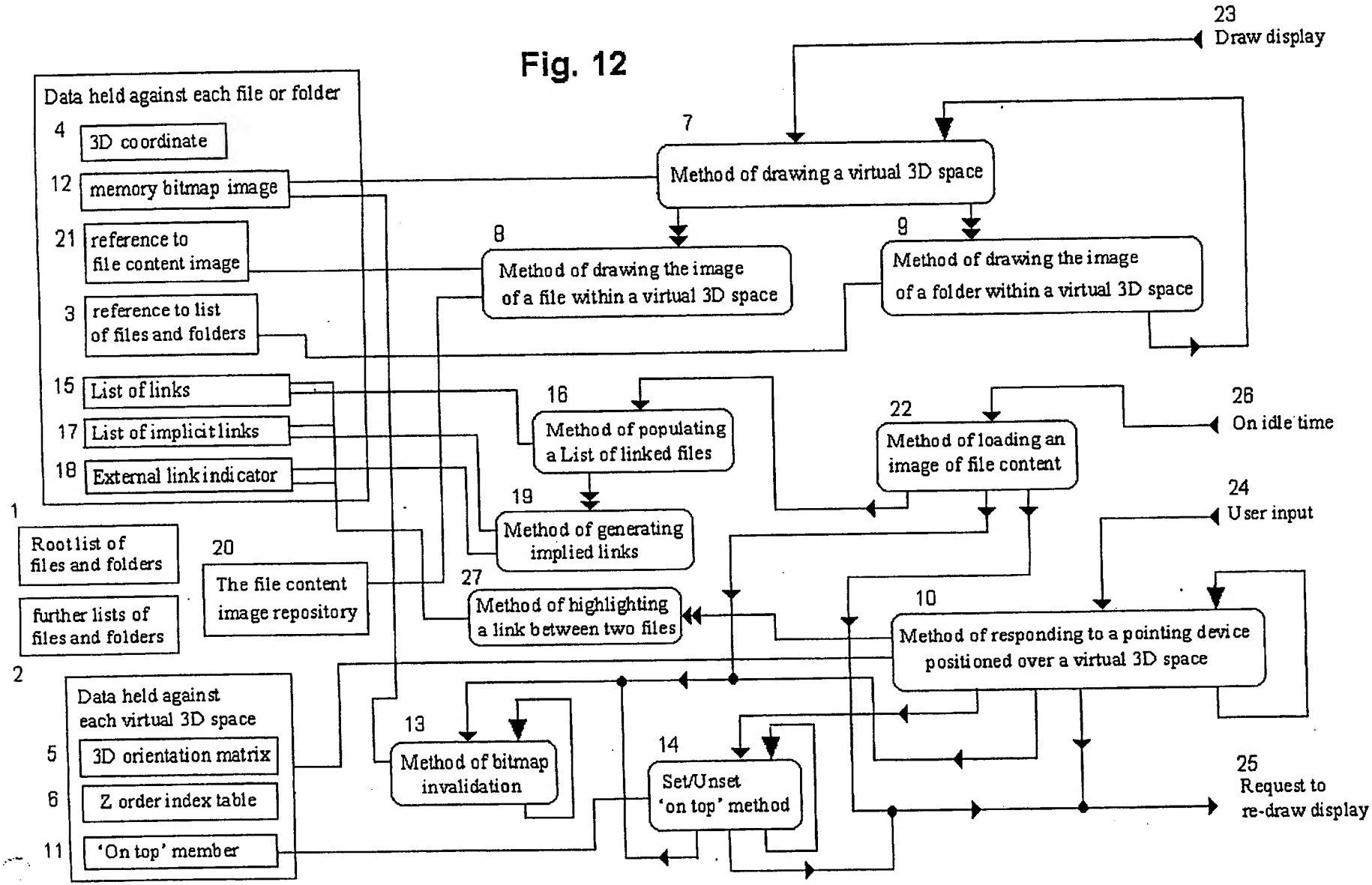

23
Draw display
Data held against each file or folder
4
3D coordinate
12
memory bitmap image
21
reference to file content image
3
reference to list of files and folders
15
List of links
17
List of implicit links
18
External link indicator
7
Method of drawing a virtual 3D space
8
Method of drawing the image of a file within a virtual 3D space
9
Method of drawing the image of a folder within a virtual 3D space
16
Method of populating a List of linked files
19
Method of generating implied links
22
Method of loading an image of file content
26
On idle time
24
User input
1
Root list of files and folders
further lists of files and folders
20
The file content image repository
27
Method of highlighting a link between two files
10
Method of responding to a pointing device positioned over a virtual 3D space
2
Data held against each virtual 3D space
5
3D orientation matrix
6
Z order index table
11
'On top' member
13
Method of bitmap invalidation
14
Set/Unset 'on top' method
25
Request to re-draw display

Fig. 13

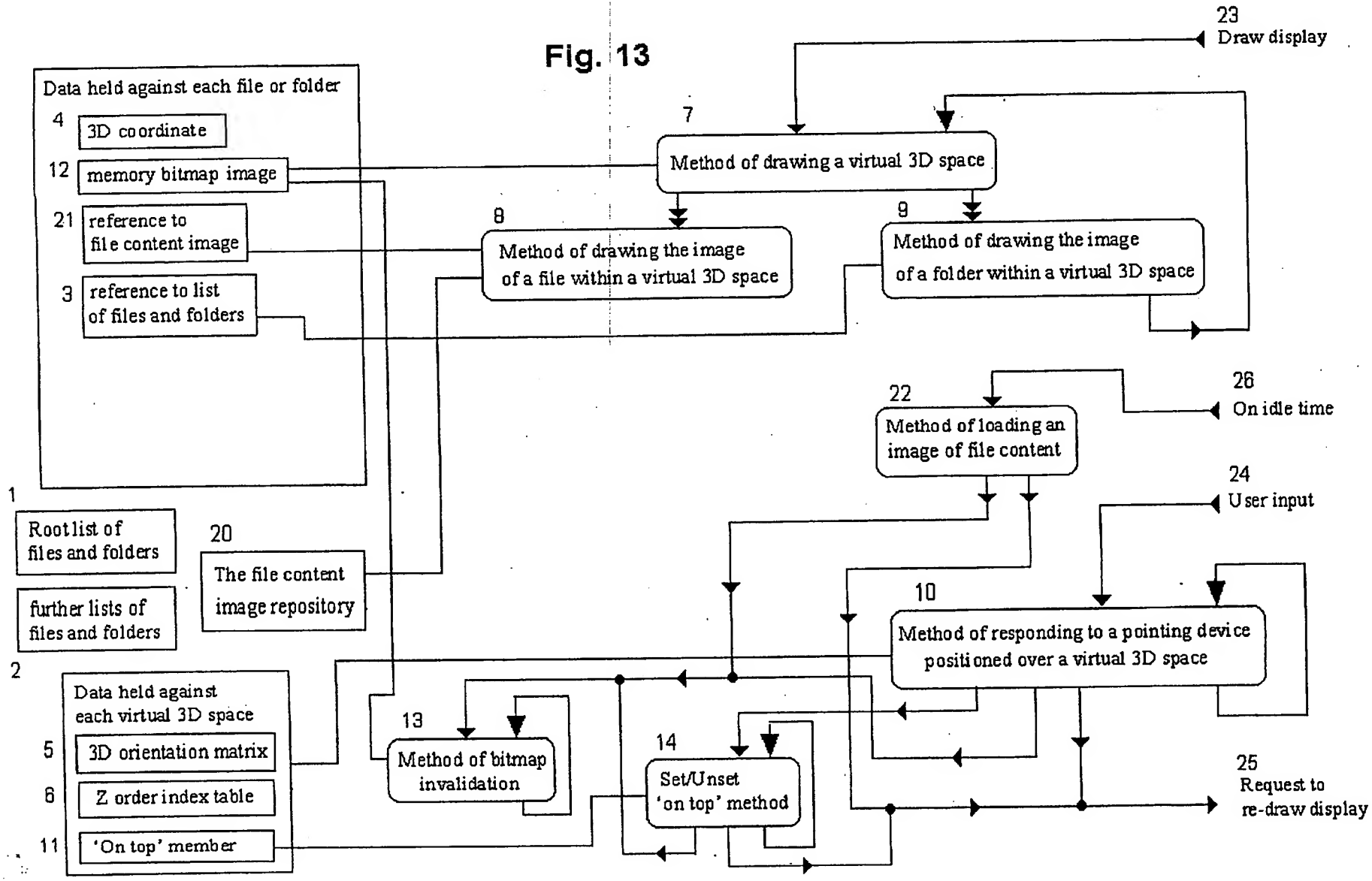

Data held against each file or folder
4
3D coordinate
12
memory bitmap image
21
reference to file content image
3
reference to list of files and folders
7
Method of drawing a virtual 3D space
8
Method of drawing the image of a file within a virtual 3D space
9
Method of drawing the image of a folder within a virtual 3D space
23
Draw display
22
Method of loading an image of file content
26
On idle time
24
User input
1
Root list of files and folders
further lists of files and folders
20
The file content image repository
10
Method of responding to a pointing device positioned over a virtual 3D space
2
Data held against each virtual 3D space
5
3D orientation matrix
6
Z order index table
11
'On top' member
13
Method of bitmap invalidation
14
Set/Unset 'on top' method
25
Request to re-draw display

Fig. 14

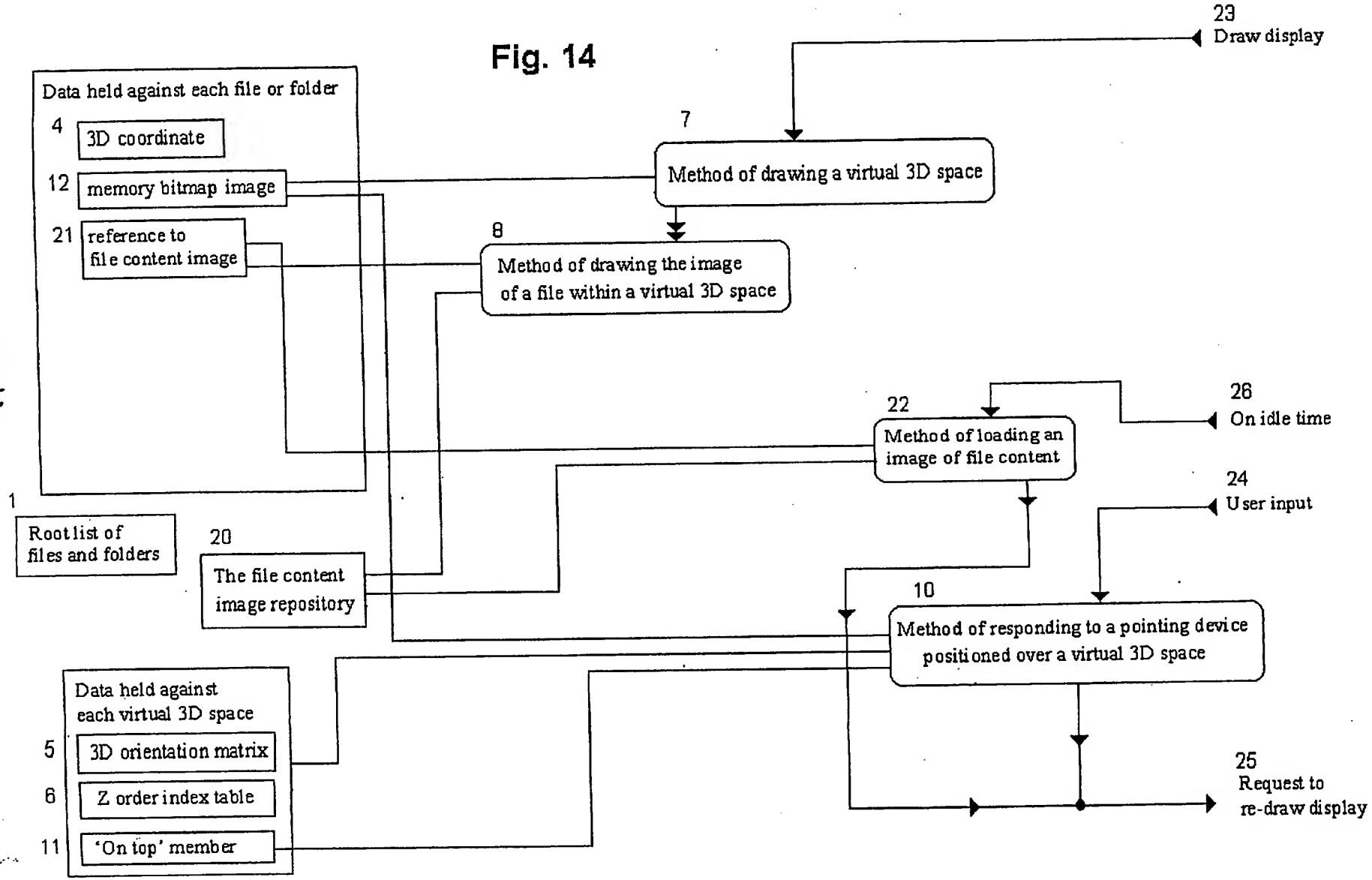

23
Draw display
Data held against each file or folder
4
3D coordinate
12
memory bitmap image
21
reference to file content image
7
Method of drawing a virtual 3D space
8
Method of drawing the image of a file within a virtual 3D space
22
Method of loading an image of file content
26
On idle time
24
User input
1
Root list of files and folders
20
The file content image repository
10
Method of responding to a pointing device positioned over a virtual 3D space
Data held against each virtual 3D space
5
3D orientation matrix
6
Z order index table
11
'On top' member
25
Request to re-draw display

Fig. 15

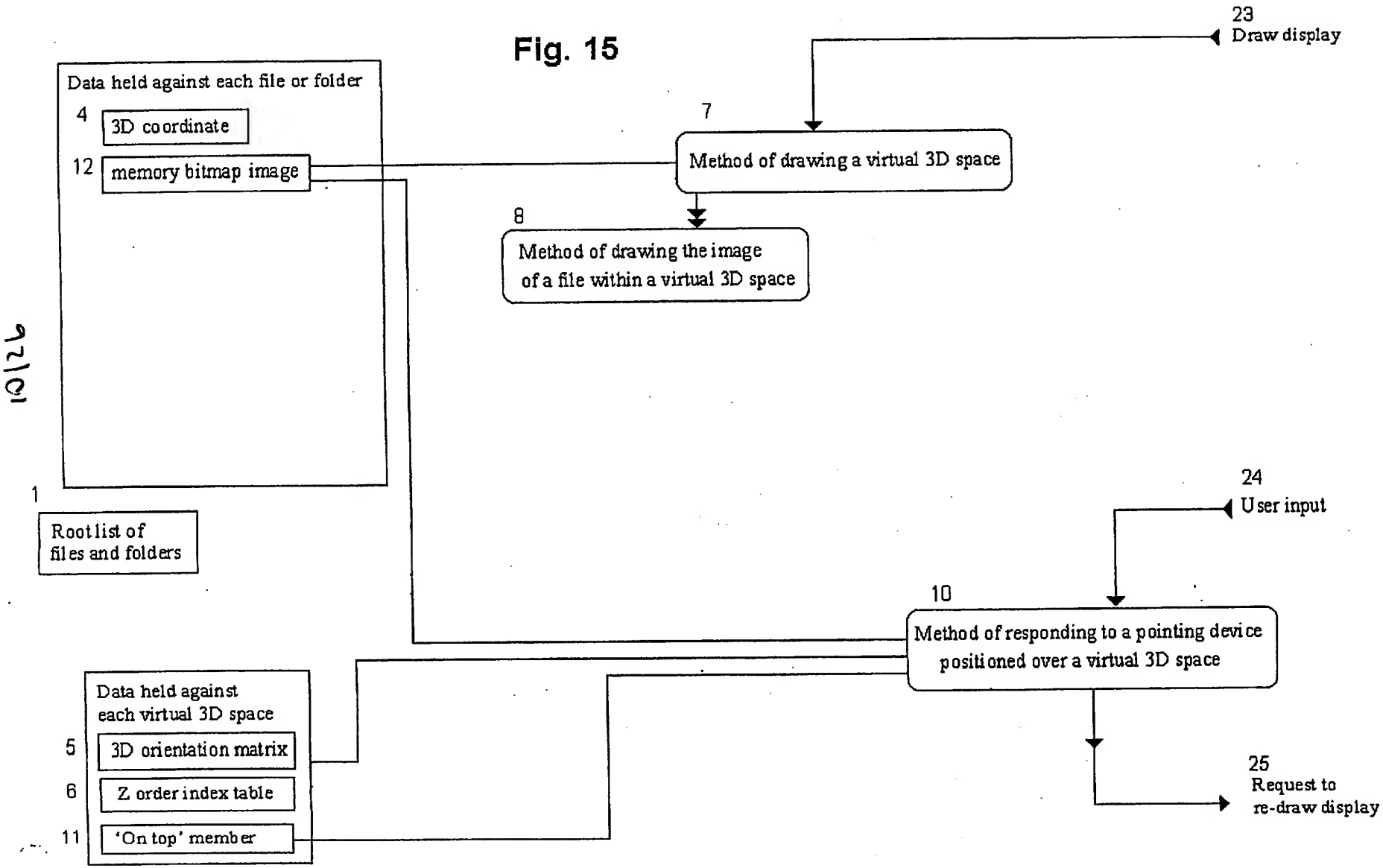

23
Draw display
Data held against each file or folder
4
3D coordinate
12
memory bitmap image
7
Method of drawing a virtual 3D space
8
Method of drawing the image of a file within a virtual 3D space
1
Root list of files and folders
24
User input
10
Method of responding to a pointing device positioned over a virtual 3D space
Data held against each virtual 3D space
5
3D orientation matrix
6
Z order index table
11
'On top' member
25
Request to re-draw display

Fig. 16

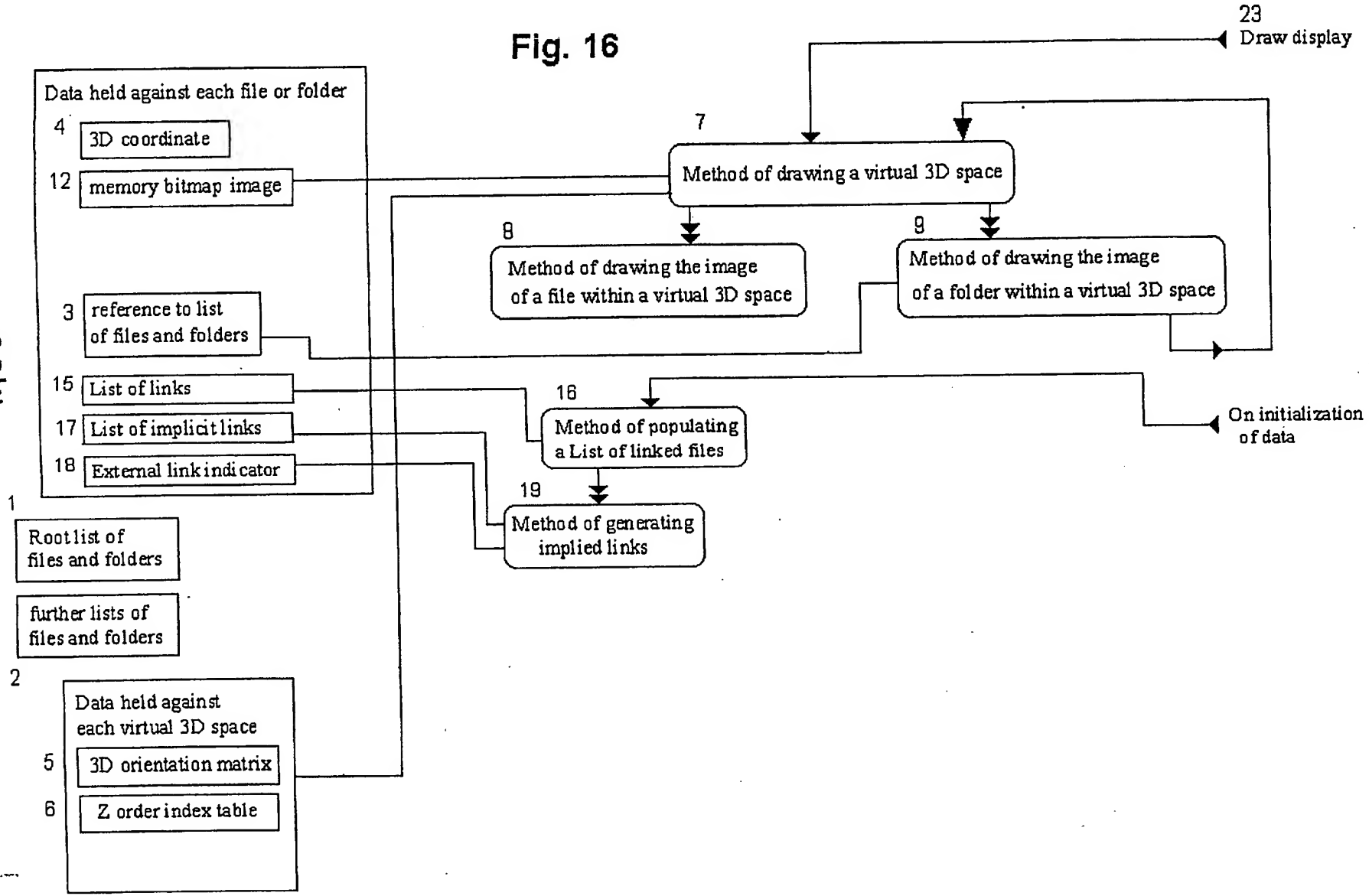
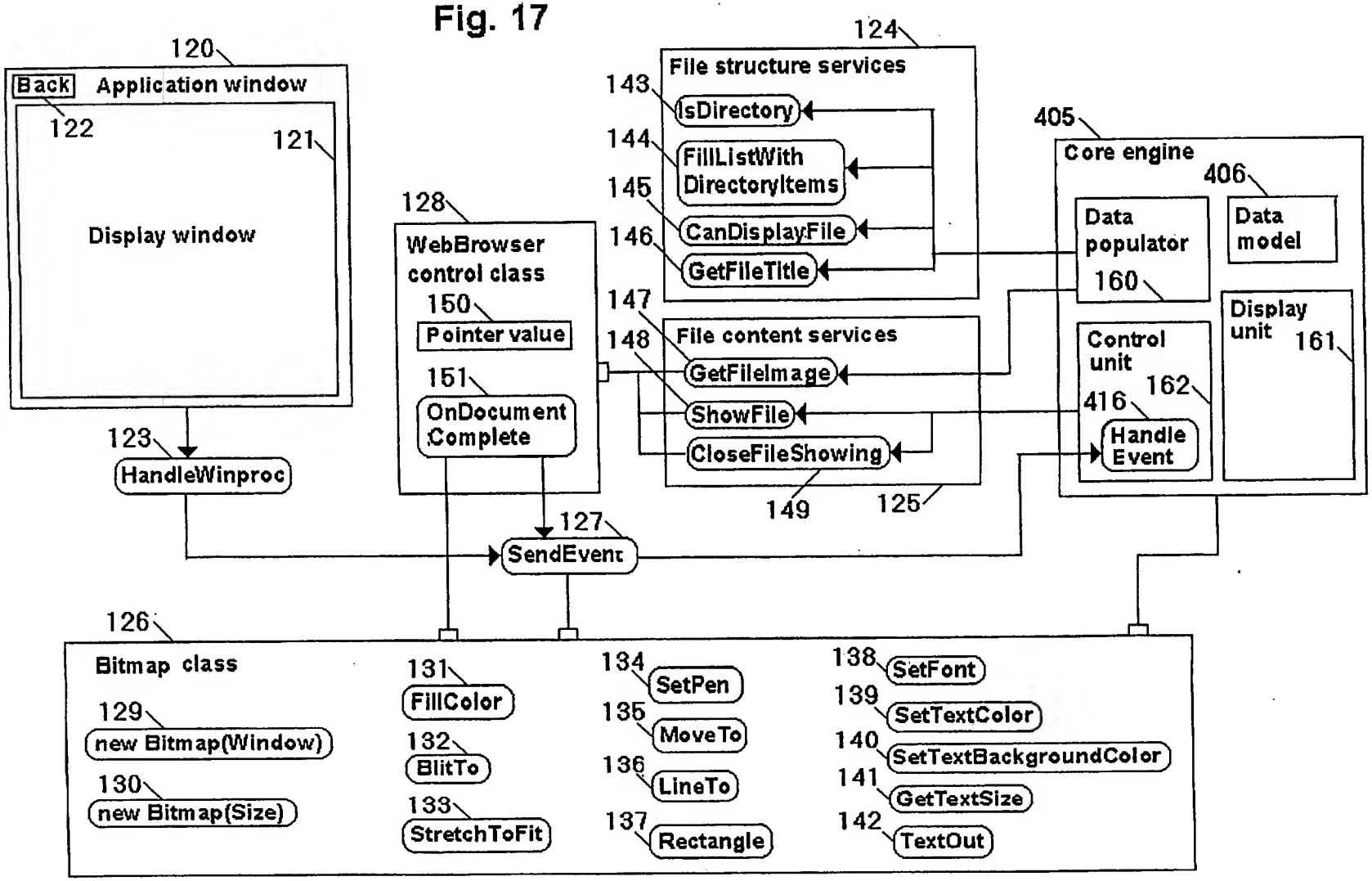

23
Draw display
Data held against each file or folder
4
3D coordinate
12
memory bitmap image
7
Method of drawing a virtual 3D space
8
Method of drawing the image of a file within a virtual 3D space
9
Method of drawing the image of a folder within a virtual 3D space
3
reference to list of files and folders
15
List of links
16
Method of populating a List of linked files
On initialization of data
17
List of implicit links
18
External link indicator
19
Method of generating implied links
1
Root list of files and folders
further lists of files and folders
2
Data held against each virtual 3D space
5
3D orientation matrix
6
Z order index table

Fig. 17


120
Back
Application window
122
121
Display window
123
HandleWinproc
128
WebBrowser control class
150
Pointer value
151
OnDocument Complete
124
File structure services
143
IsDirectory
144
FillListWith DirectoryItems
145
CanDisplayFile
146
GetFileTitle
147
File content services
148
GetFileImage
ShowFile
CloseFileShowing
149
125
127
SendEvent
405
Core engine
406
Data populator
Data model
160
Control unit
162
416
Handle Event
Display unit
161
126
Bitmap class
129
new Bitmap(Window)
130
new Bitmap(Size)
131
FillColor
132
BlitTo
133
StretchToFit
134
SetPen
135
MoveTo
136
LineTo
137
Rectangle
138
SetFont
139
SetTextColor
140
SetTextBackgroundColor
141
GetTextSize
142
TextOut

12/26

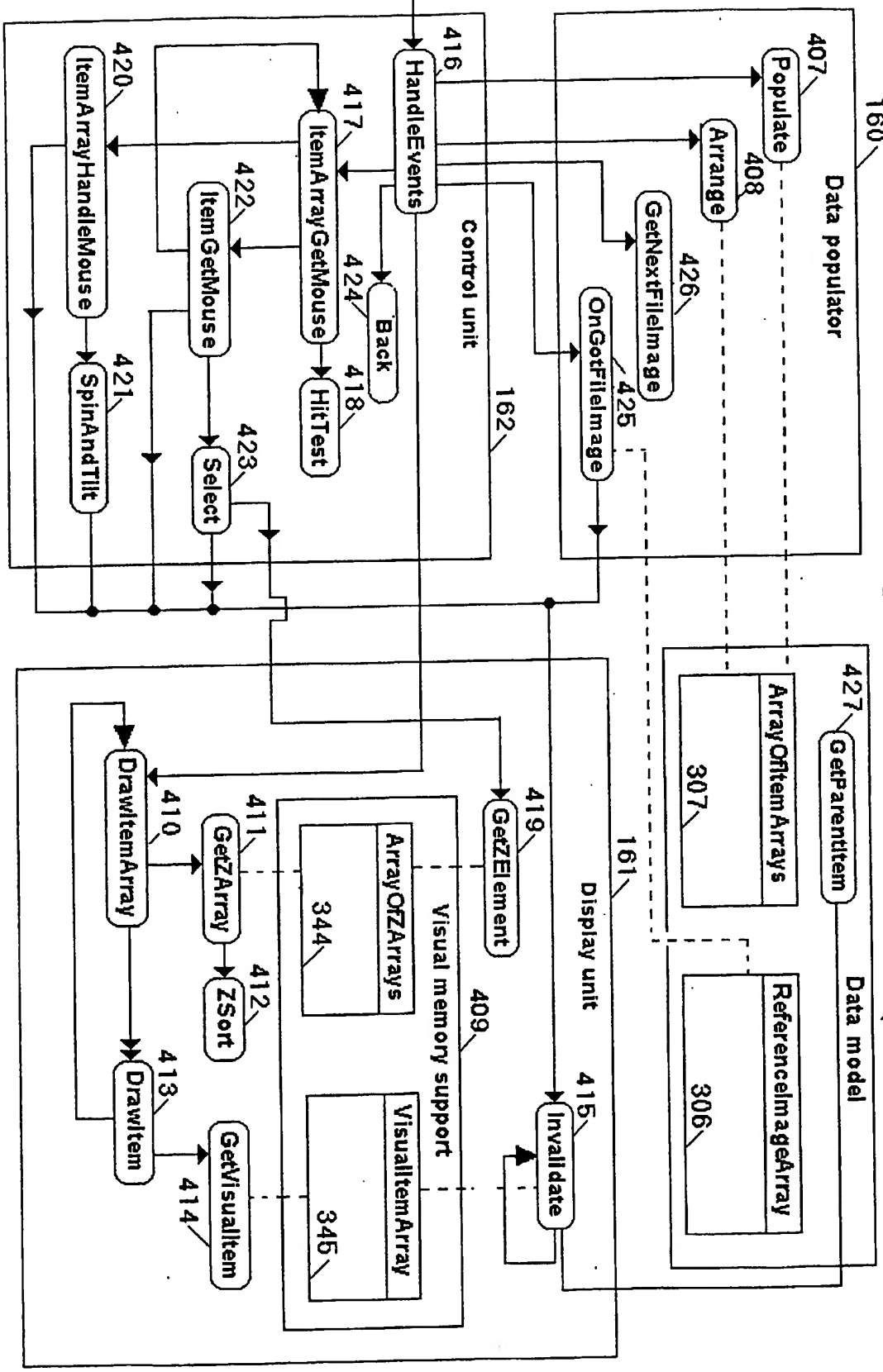
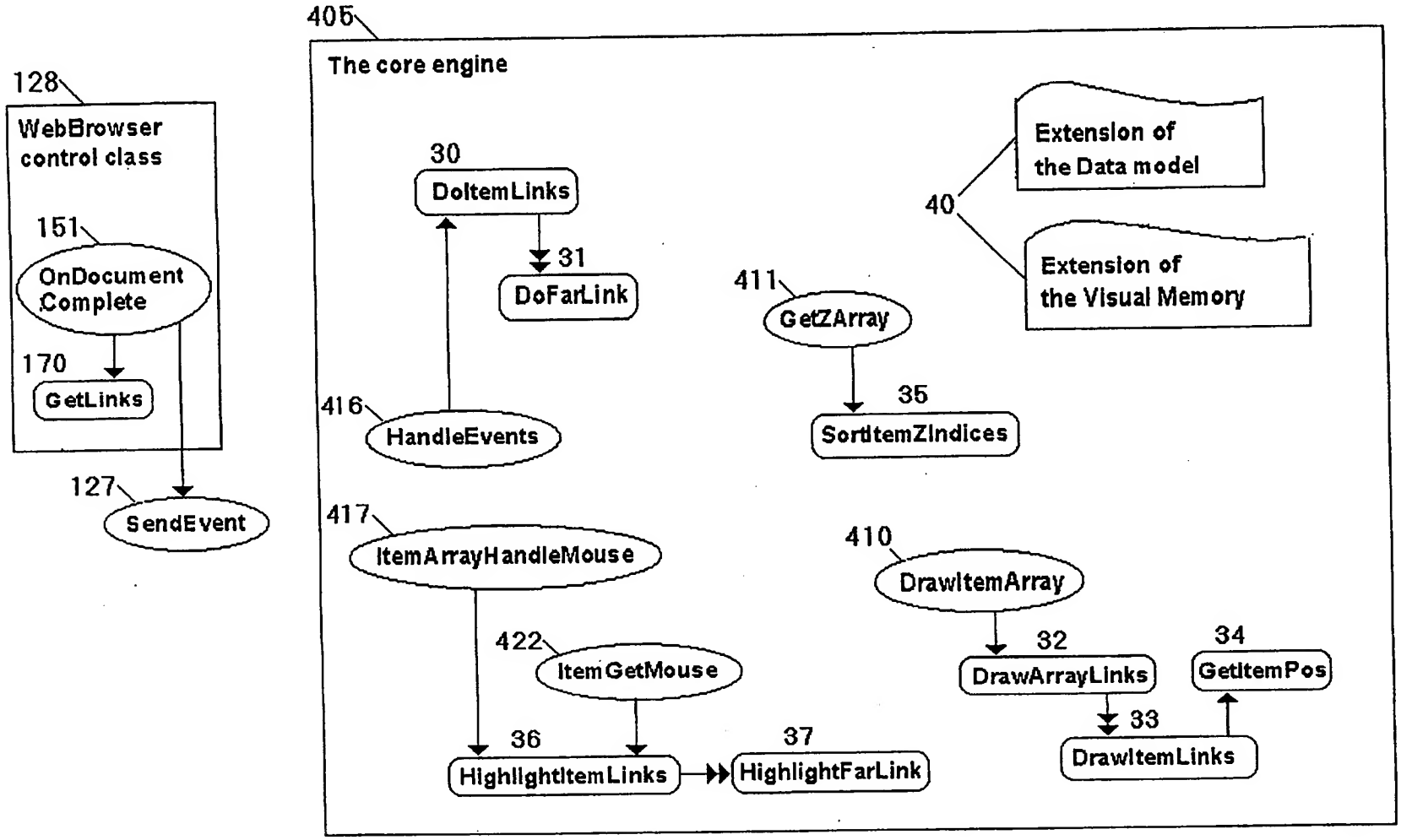

160
Data populator
407
Populate
408
Arrange
426
GetNextFileImage
425
OnGotFileImage
416
HandleEvents
417
ItemArrayGetMouse
424
Back
418
HitTest
422
ItemGetMouse
423
Select
420
ItemArrayHandleMouse
421
SpinAndTilt
Control unit
162
406
Data model
427
GetParentItem
ArrayOfItemArrays
307
ReferenceImageArray
306
161
Display unit
415
Invalidate
419
GetZElement
409
Visual memory support
ArrayOfZArrays
344
VisualItemArray
345
410
DrawItemArray
411
GetZArray
412
ZSort
413
DrawItem
414
GetVisualItem

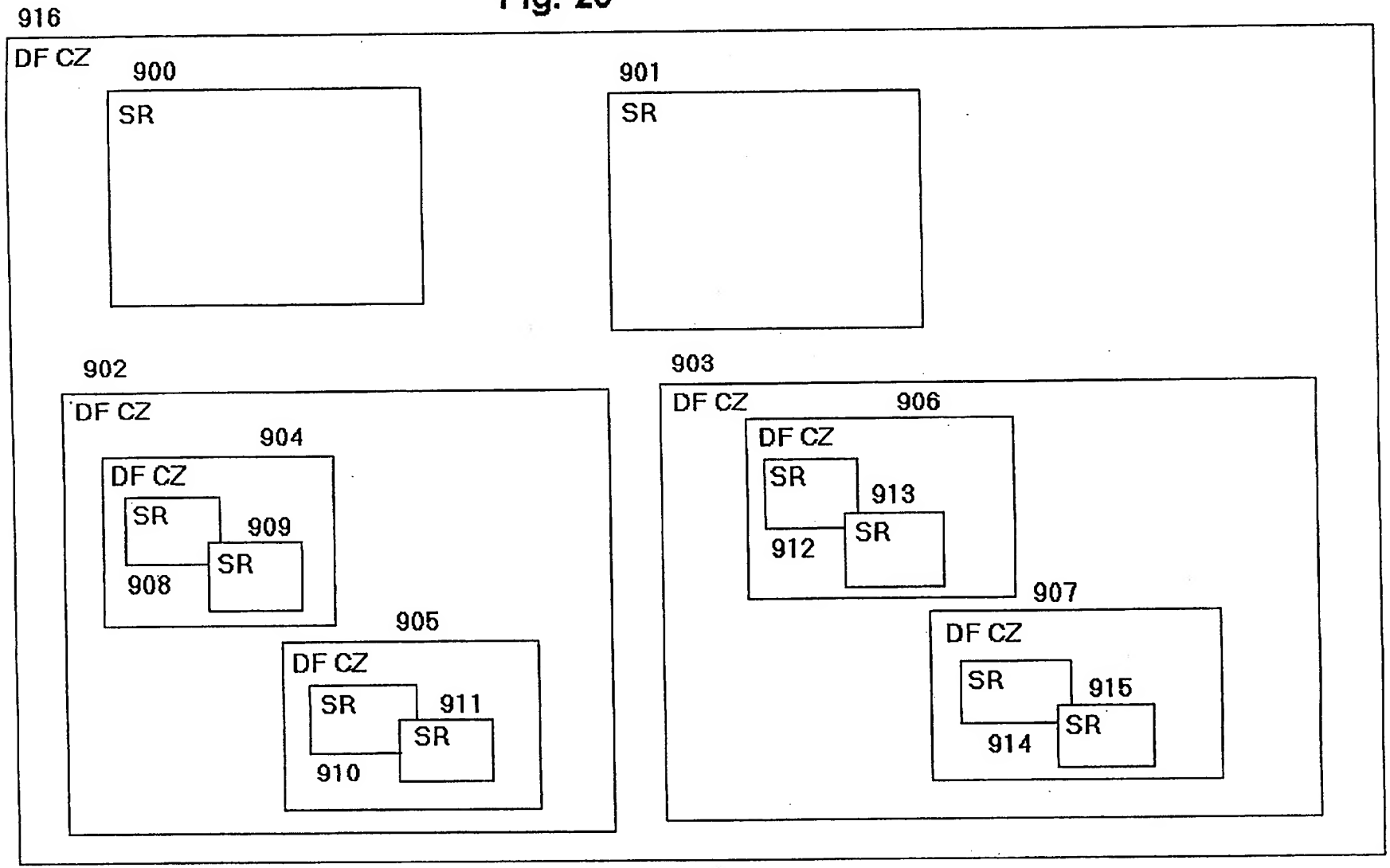
Fig. 18

Fig. 19


128
WebBrowser control class
151
OnDocument Complete
170
GetLinks
127
SendEvent
405
The core engine
30
DoItemLinks
31
DoFarLink
416
HandleEvents
411
GetZArray
35
SortItemZIndices
40
Extension of the Data model
Extension of the Visual Memory
417
ItemArrayHandleMouse
422
ItemGetMouse
36
HighlightItemLinks
37
HighlightFarLink
410
DrawItemArray
32
DrawArrayLinks
34
GetItemPos
33
DrawItemLinks

1426

Fig. 20


916
DF CZ
900
SR
901
SR
902
DF CZ
904
DF CZ
SR
909
SR
908
905
DF CZ
SR
911
SR
910
903
DF CZ
906
DF CZ
SR
913
SR
912
907
DF CZ
SR
915
SR
914

15126

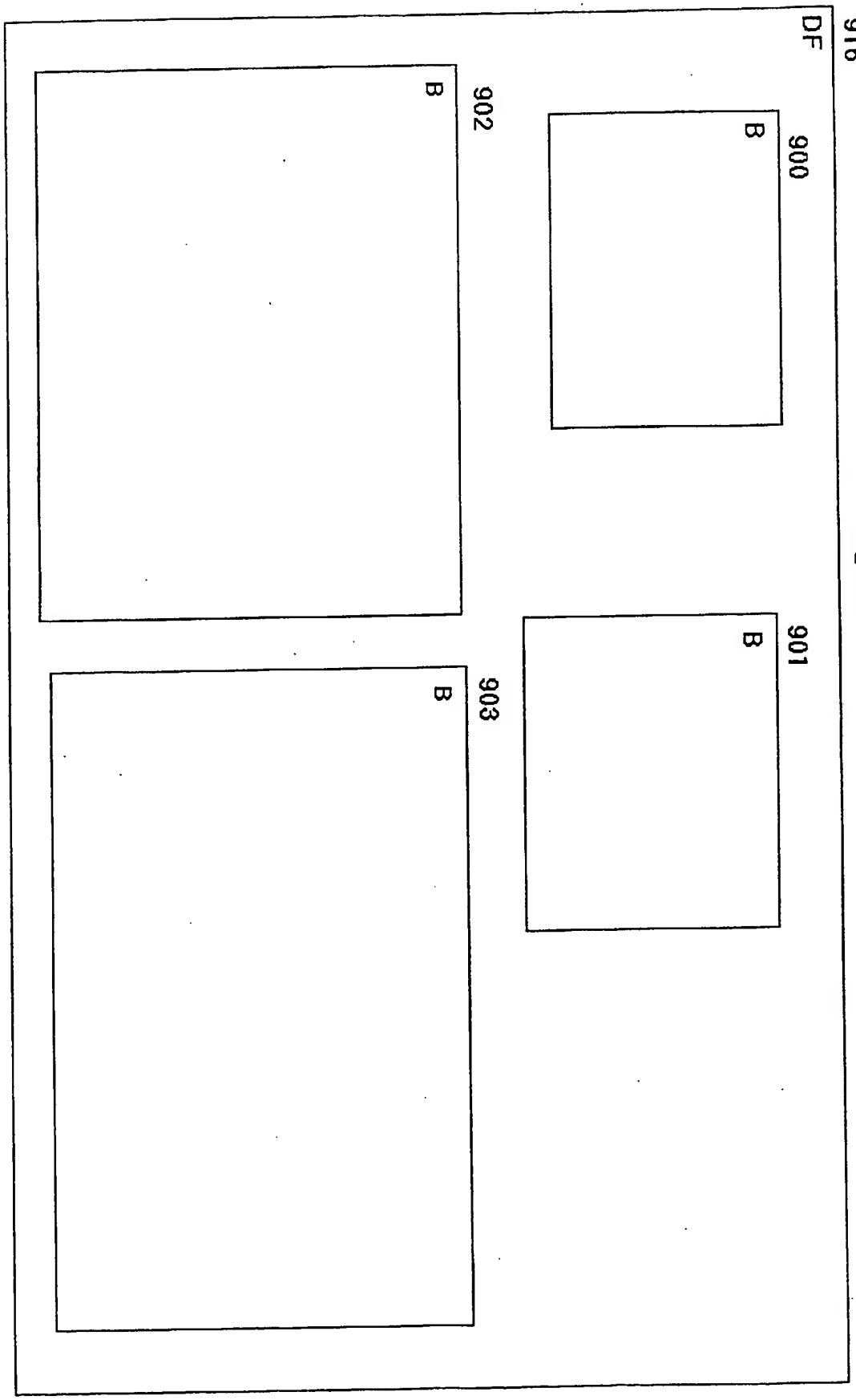
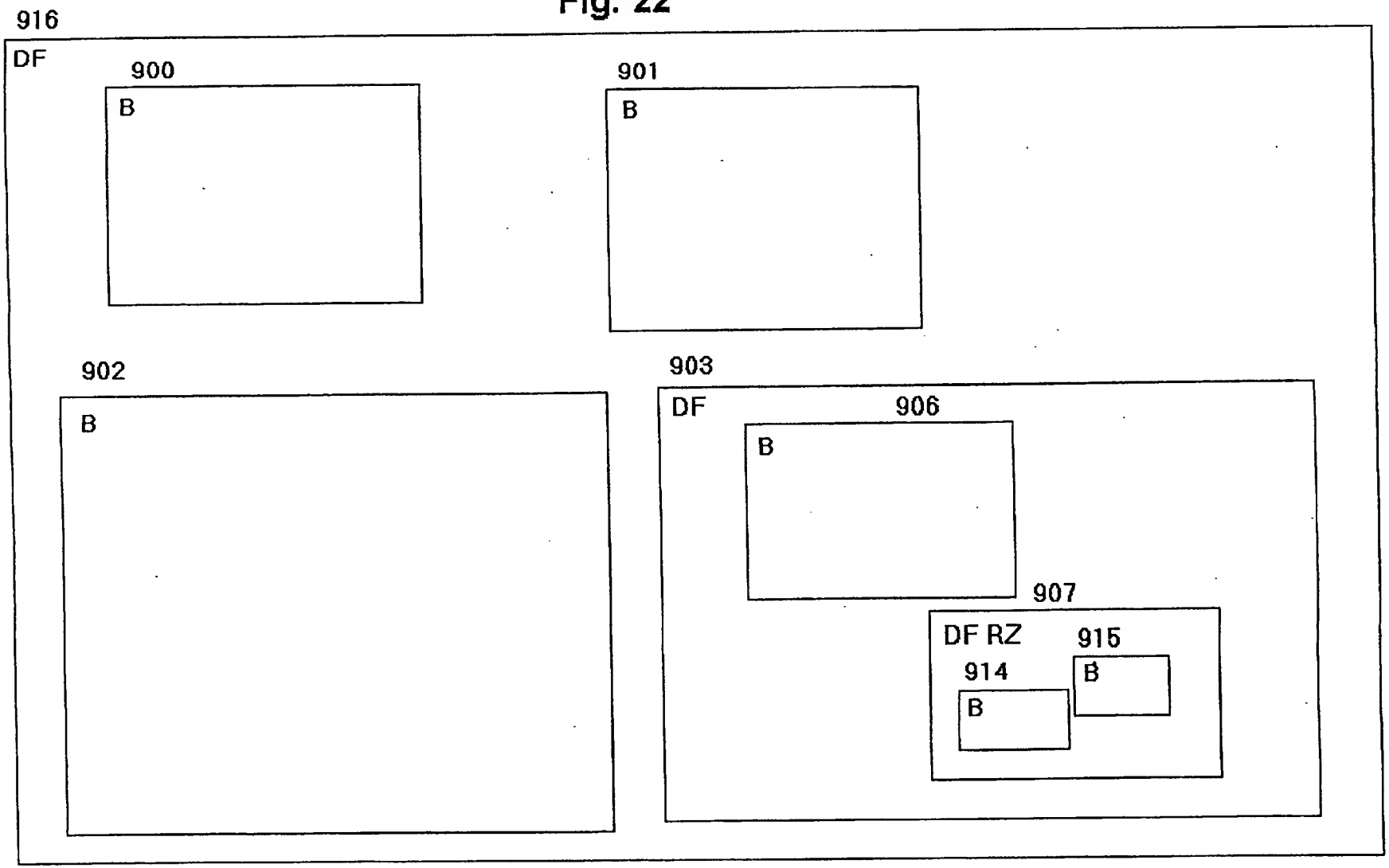


Fig. 21

Fig. 22



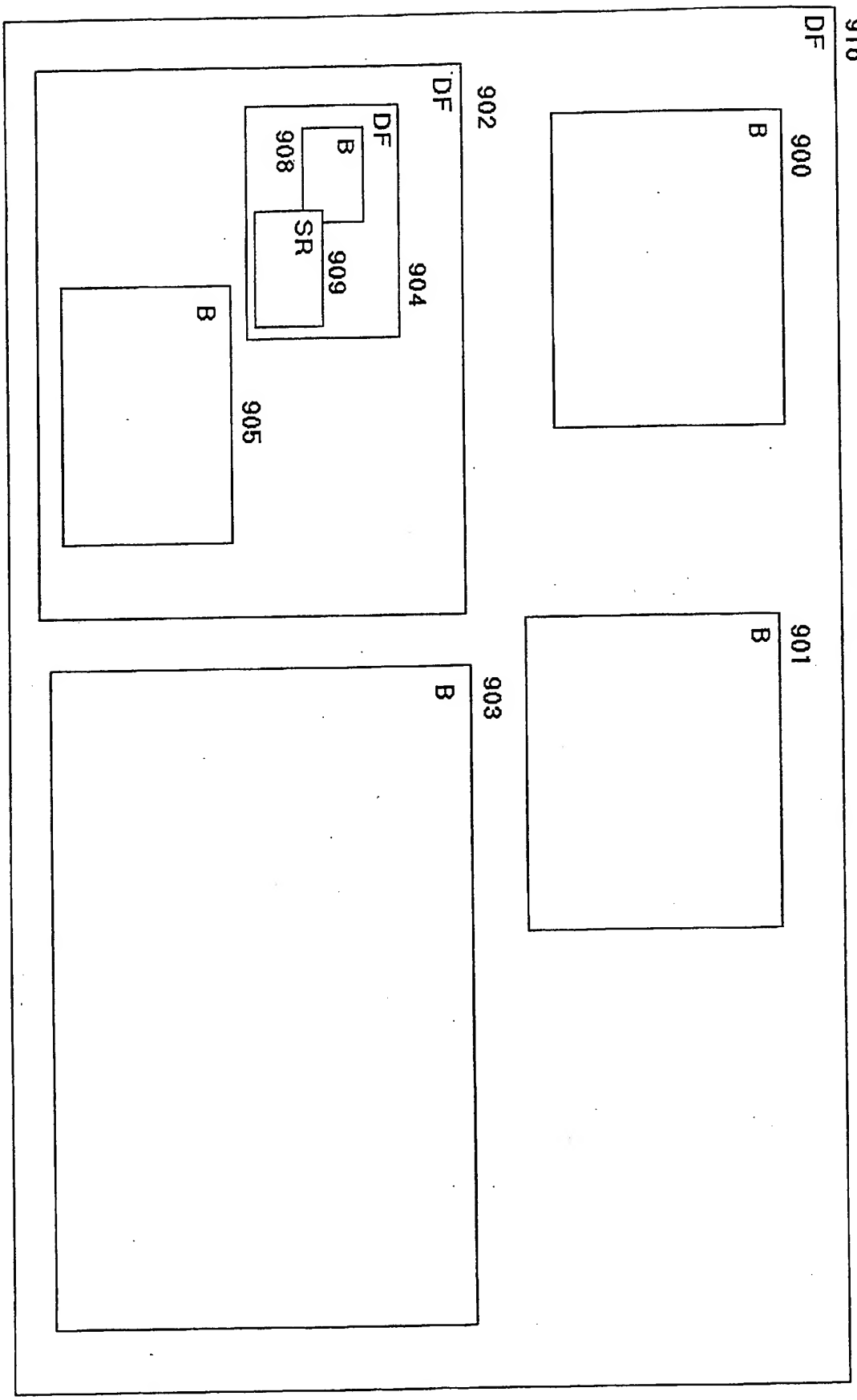


Fig. 23

19126

Fig. 24

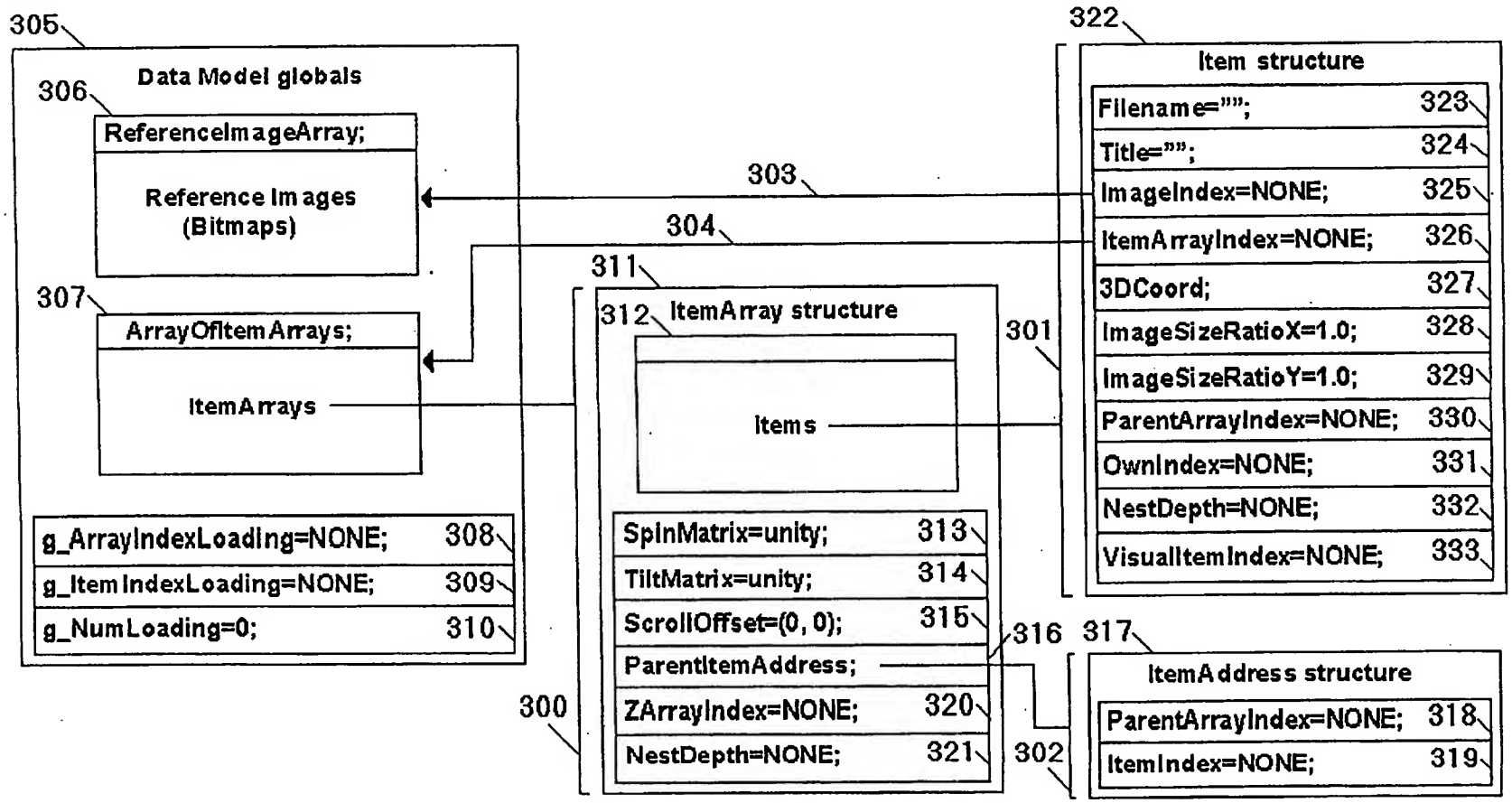

305
Data Model globals
306
ReferenceImageArray;
Reference Images
(Bitmaps)
307
ArrayOfItemArrays;
ItemArrays
g_ArrayIndexLoading=NONE; 308
g_ItemIndexLoading=NONE; 309
g_NumLoading=0; 310
303
304
311
312 ItemArray structure
Items
SpinMatrix=unity; 313
TiltMatrix=unity; 314
ScrollOffset=(0, 0); 315
ParentItemAddress; 316
ZArrayIndex=NONE; 320
NestDepth=NONE; 321
300
301
322
Item structure
Filename=""; 323
Title=""; 324
ImageIndex=NONE; 325
ItemArrayIndex=NONE; 326
3DCoord; 327
ImageSizeRatioX=1.0; 328
ImageSizeRatioY=1.0; 329
ParentArrayIndex=NONE; 330
OwnIndex=NONE; 331
NestDepth=NONE; 332
VisualItemIndex=NONE; 333
317
ItemAddress structure
ParentArrayIndex=NONE; 318
ItemIndex=NONE; 319
302

Fig. 25

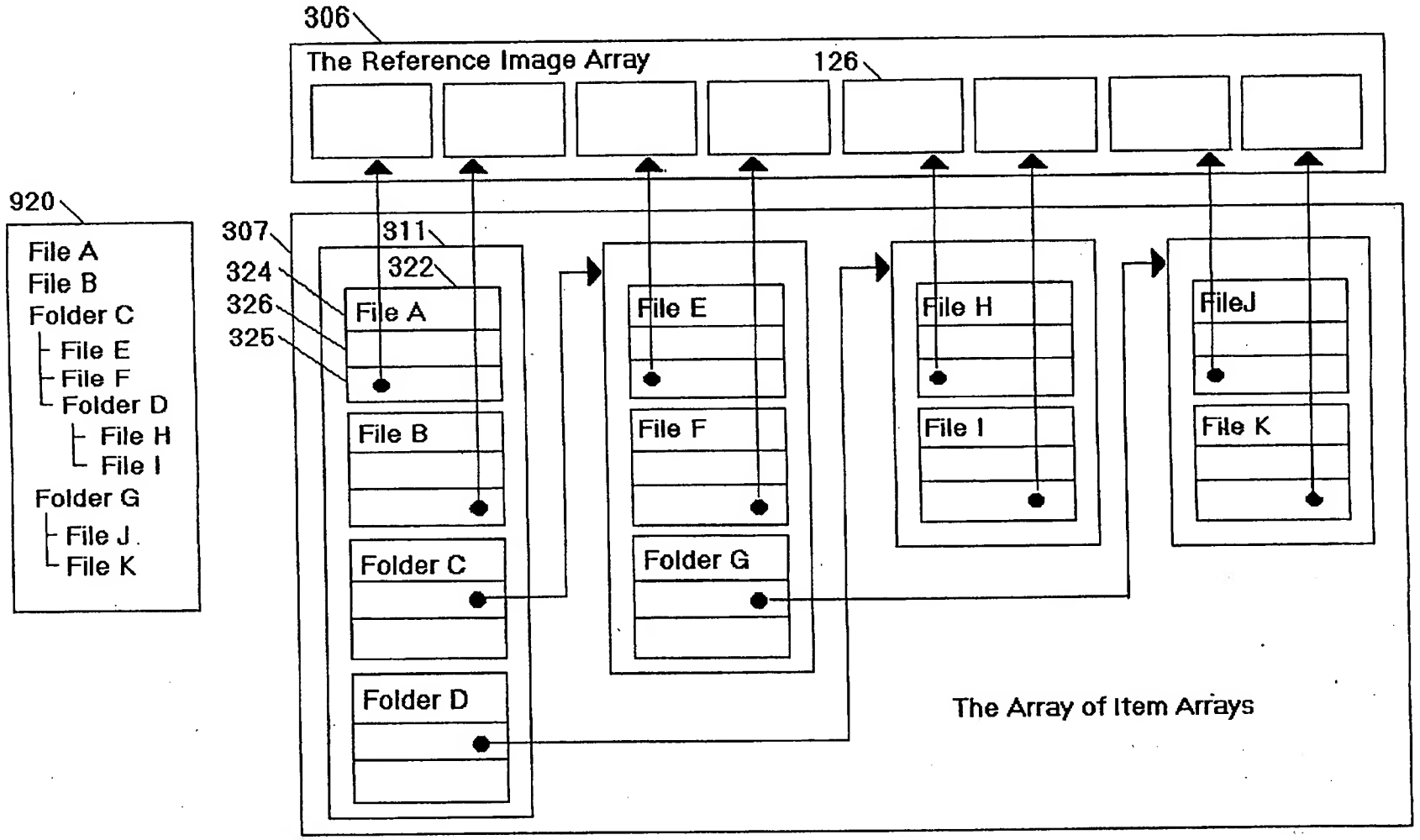
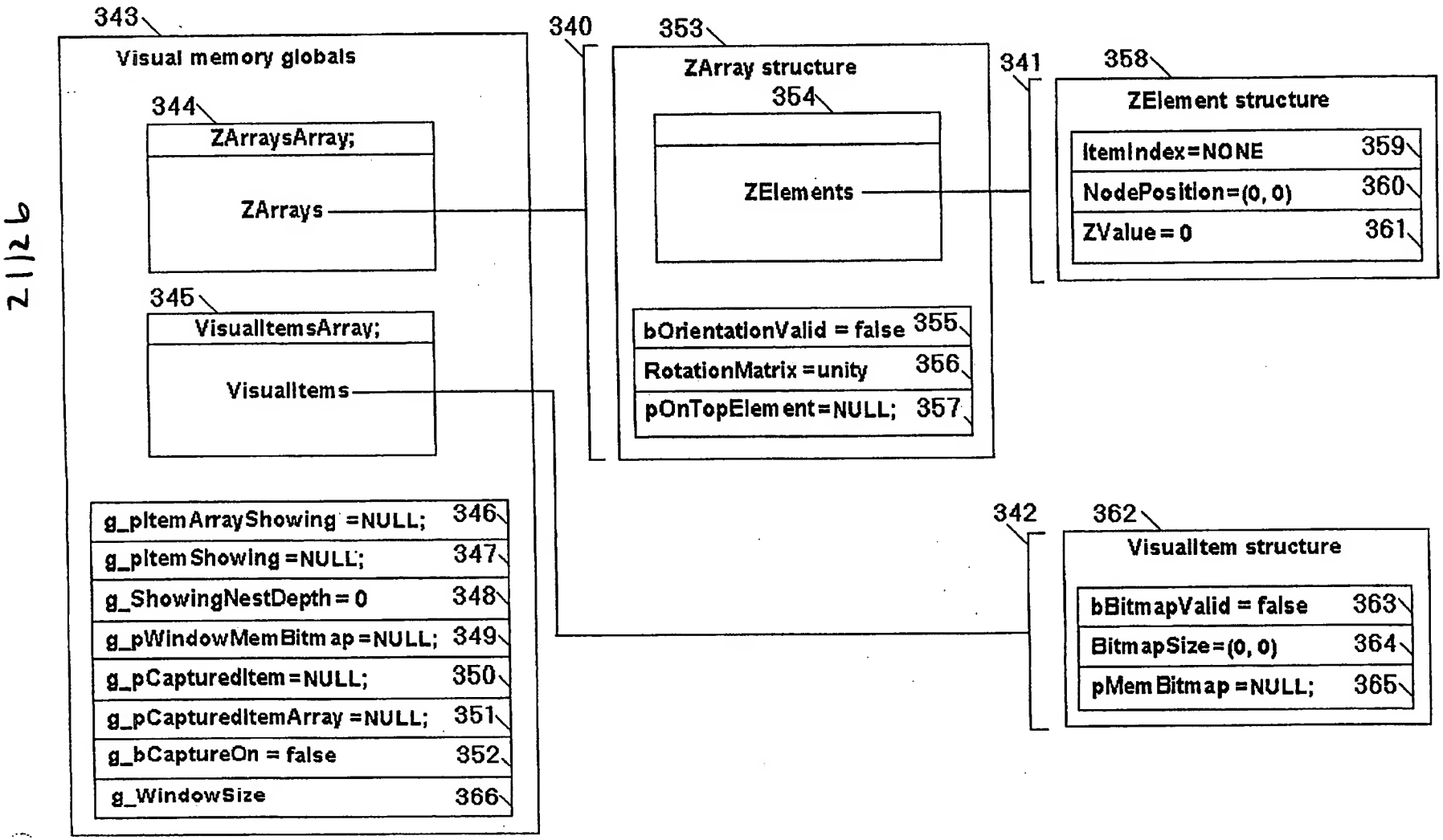

306
The Reference Image Array
126
920
File A
File B
Folder C
File E
File F
Folder D
File H
File I
Folder G
File J
File K
307
311
322
324
326
325
File A
File B
Folder C
Folder D
File E
File F
Folder G
File H
File I
FileJ
File K
The Array of Item Arrays

Fig. 26


343
Visual memory globals
344
ZArraysArray;
ZArrays
345
VisualItemsArray;
VisualItems
g_pItemArrayShowing =NULL; 346
g_pItemShowing =NULL; 347
g_ShowingNestDepth = 0 348
g_pWindowMemBitmap =NULL; 349
g_pCapturedItem =NULL; 350
g_pCapturedItemArray =NULL; 351
g_bCaptureOn = false 352
g_WindowSize 366
340
353
ZArray structure
354
ZElements
bOrientationValid = false 355
RotationMatrix =unity 356
pOnTopElement=NULL; 357
341
358
ZElement structure
ItemIndex=NONE 359
NodePosition=(0, 0) 360
ZValue = 0 361
342
362
VisualItem structure
bBitmapValid = false 363
BitmapSize=(0, 0) 364
pMemBitmap =NULL; 365

2126

Fig. 27

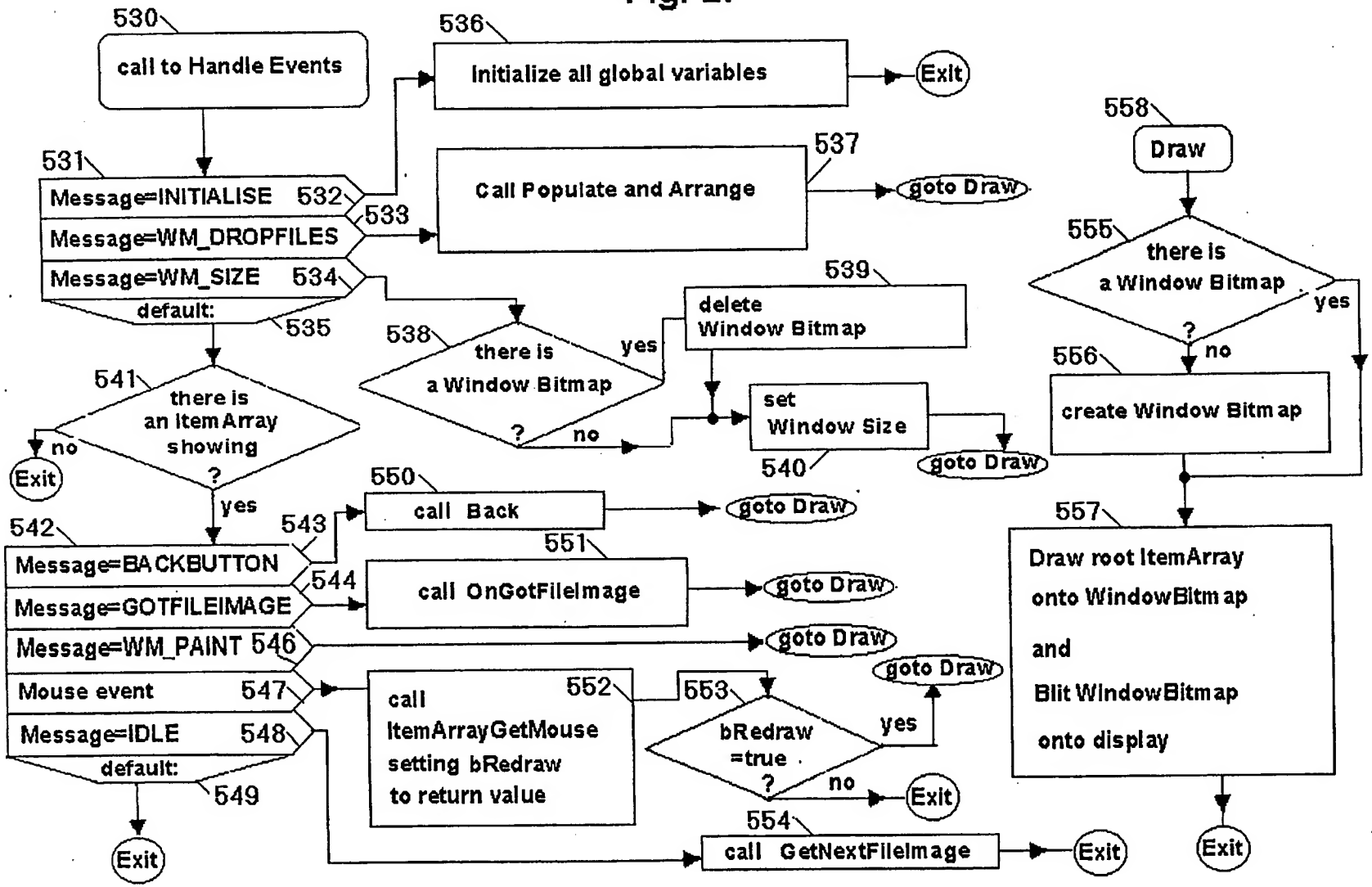

530
call to Handle Events
536
Initialize all global variables
Exit
558
Draw
531
Message=INITIALISE 532
Message=WM_DROPFILES 533
Message=WM_SIZE 534
default: 535
Call Populate and Arrange
537
goto Draw
555
there is a Window Bitmap ?
yes
no
539
delete Window Bitmap
538
there is a Window Bitmap ?
yes
no
556
create Window Bitmap
541
there is an ItemArray showing ?
no
Exit
yes
set Window Size
540
goto Draw
542
Message=BACKBUTTON
Message=GOTFILEIMAGE
Message=WM_PAINT 546
Mouse event 547
Message=IDLE 548
default: 549
543
550
call Back
goto Draw
544
551
call OnGotFileImage
goto Draw
goto Draw
552
call ItemArrayGetMouse setting bRedraw to return value
553
bRedraw =true ?
yes
no
goto Draw
Exit
554
call GetNextFileImage
Exit
557
Draw root ItemArray onto WindowBitmap and Blit WindowBitmap onto display
Exit

Fig. 28

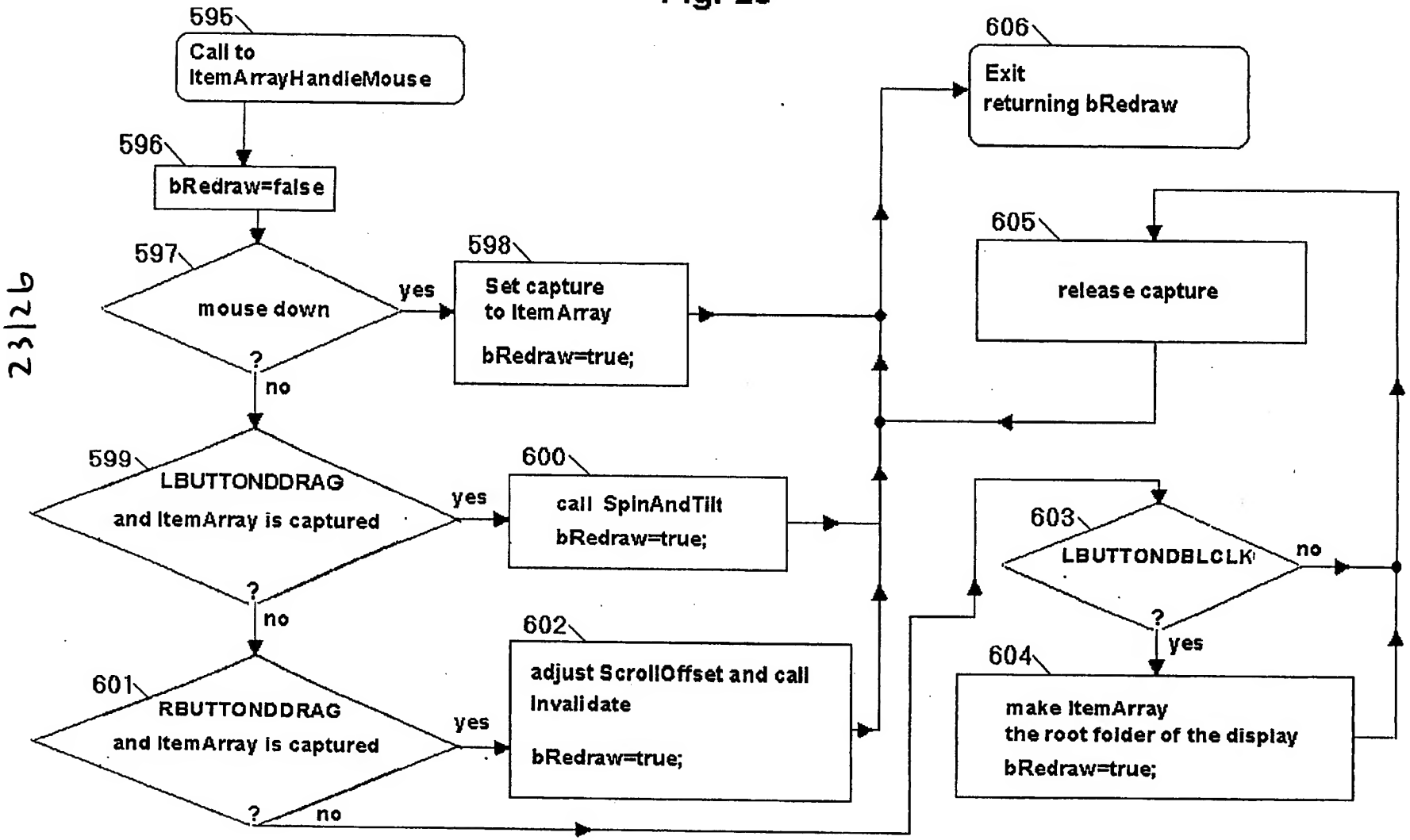

595
Call to
ItemArrayHandleMouse
596
bRedraw=false
597
mouse down
?
yes
no
598
Set capture
to ItemArray
bRedraw=true;
599
LBUTTONDDRAG
and ItemArray is captured
?
yes
no
600
call SpinAndTilt
bRedraw=true;
601
RBUTTONDDRAG
and ItemArray is captured
?
yes
no
602
adjust ScrollOffset and call
Invalidate
bRedraw=true;
603
LBUTTONDBLCLK
?
no
yes
604
make ItemArray
the root folder of the display
bRedraw=true;
605
release capture
606
Exit
returning bRedraw

Fig. 29

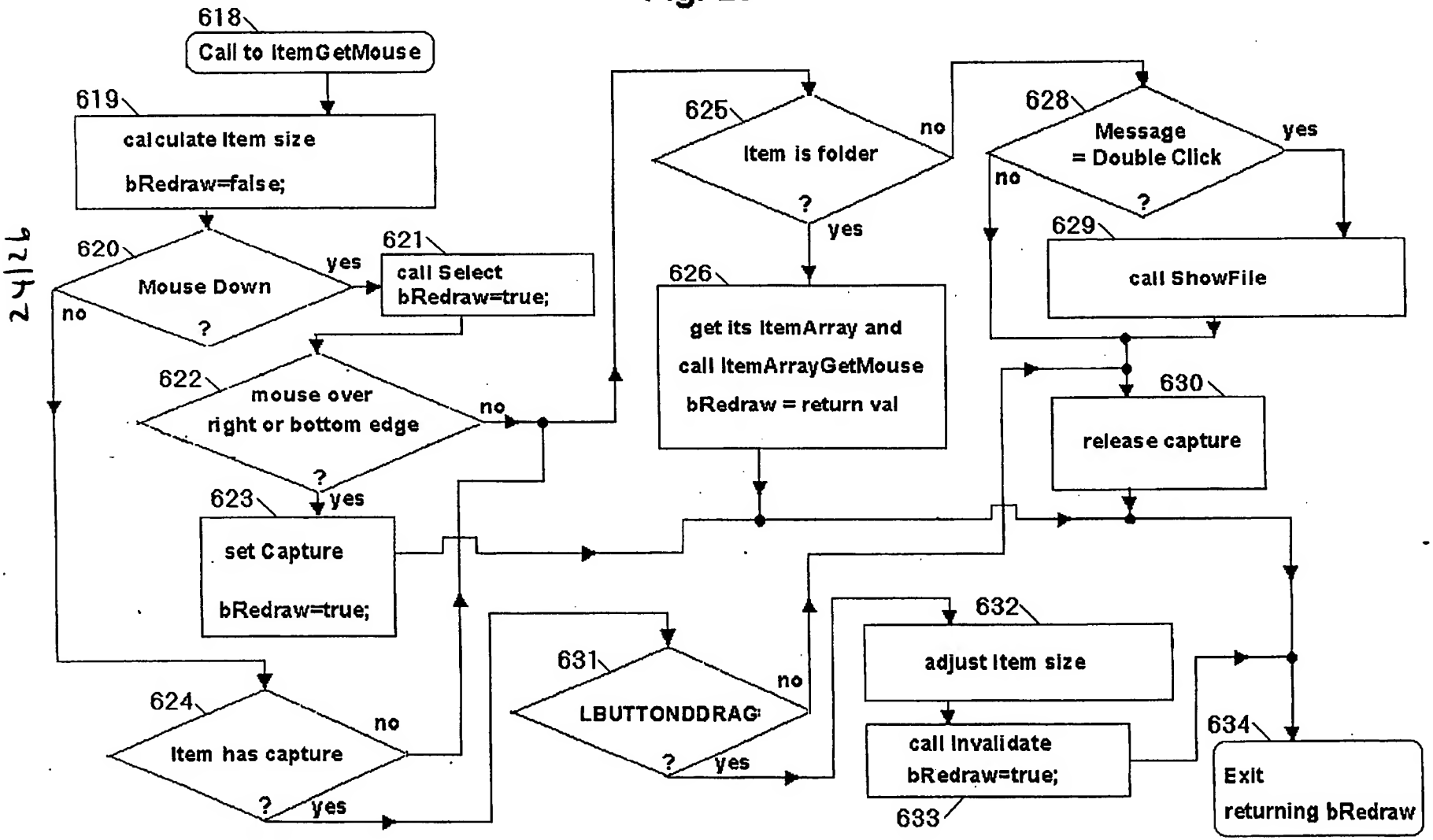
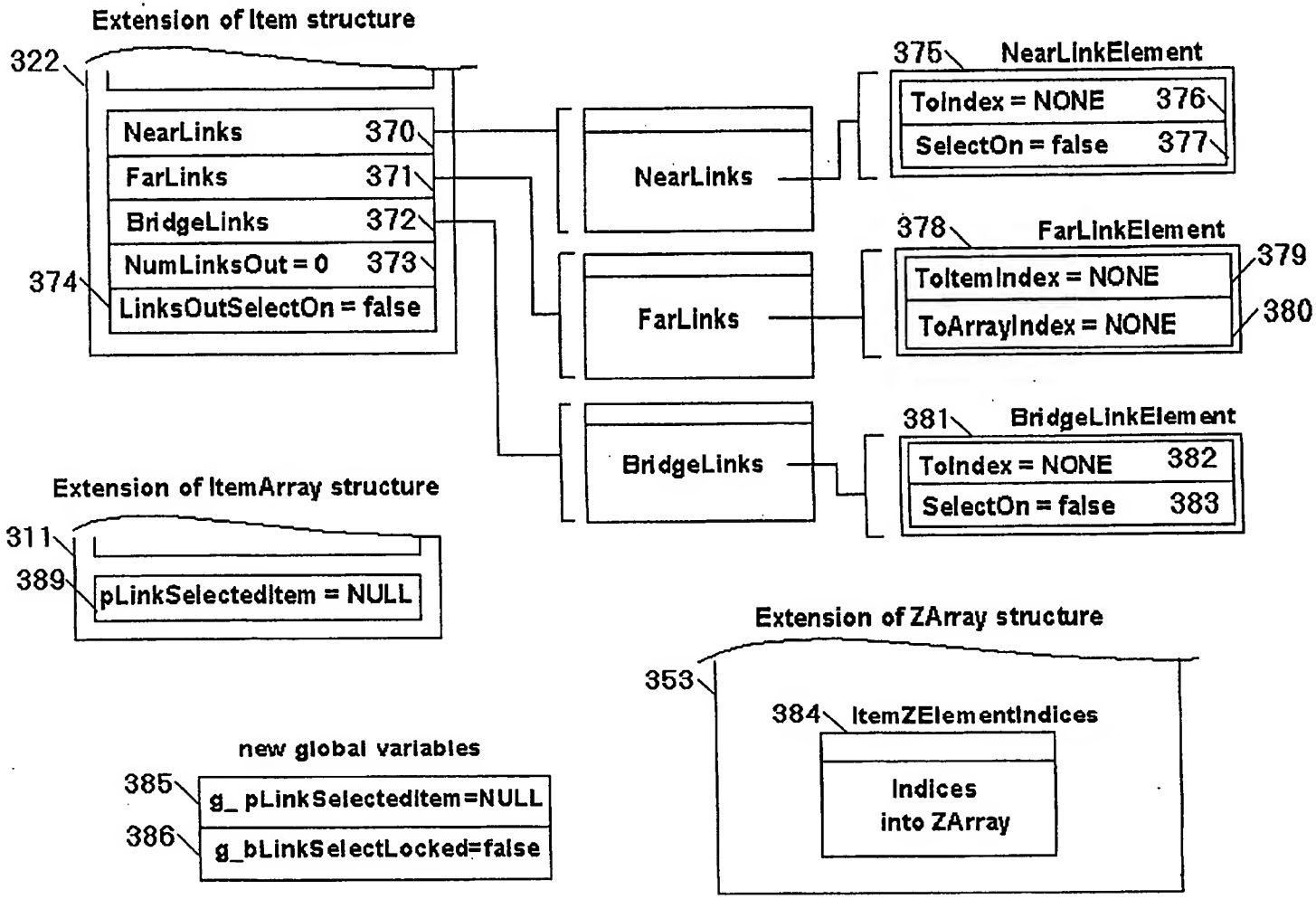

618
Call to ItemGetMouse
619
calculate item size
bRedraw=false;
620
Mouse Down
?
yes
no
621
call Select
bRedraw=true;
622
mouse over
right or bottom edge
?
no
yes
623
set Capture
bRedraw=true;
624
Item has capture
?
no
yes
625
Item is folder
?
no
yes
626
get its ItemArray and
call ItemArrayGetMouse
bRedraw = return val
628
Message
= Double Click
?
yes
no
629
call ShowFile
630
release capture
631
LBUTTONDRAG
?
no
yes
632
adjust item size
call Invalidate
bRedraw=true;
633
634
Exit
returning bRedraw

Fig. 30


Extension of Item structure
322
NearLinks 370
FarLinks 371
BridgeLinks 372
NumLinksOut = 0 373
374
LinksOutSelectOn = false
375 NearLinkElement
ToIndex = NONE 376
SelectOn = false 377
NearLinks
378 FarLinkElement
ToItemIndex = NONE 379
ToArrayIndex = NONE 380
FarLinks
381 BridgeLinkElement
ToIndex = NONE 382
SelectOn = false 383
BridgeLinks
Extension of ItemArray structure
311
389
pLinkSelectedItem = NULL
Extension of ZArray structure
353
384 ItemZElementIndices
Indices into ZArray
new global variables
385
g_ pLinkSelectedItem=NULL
386
g_bLinkSelectLocked=false

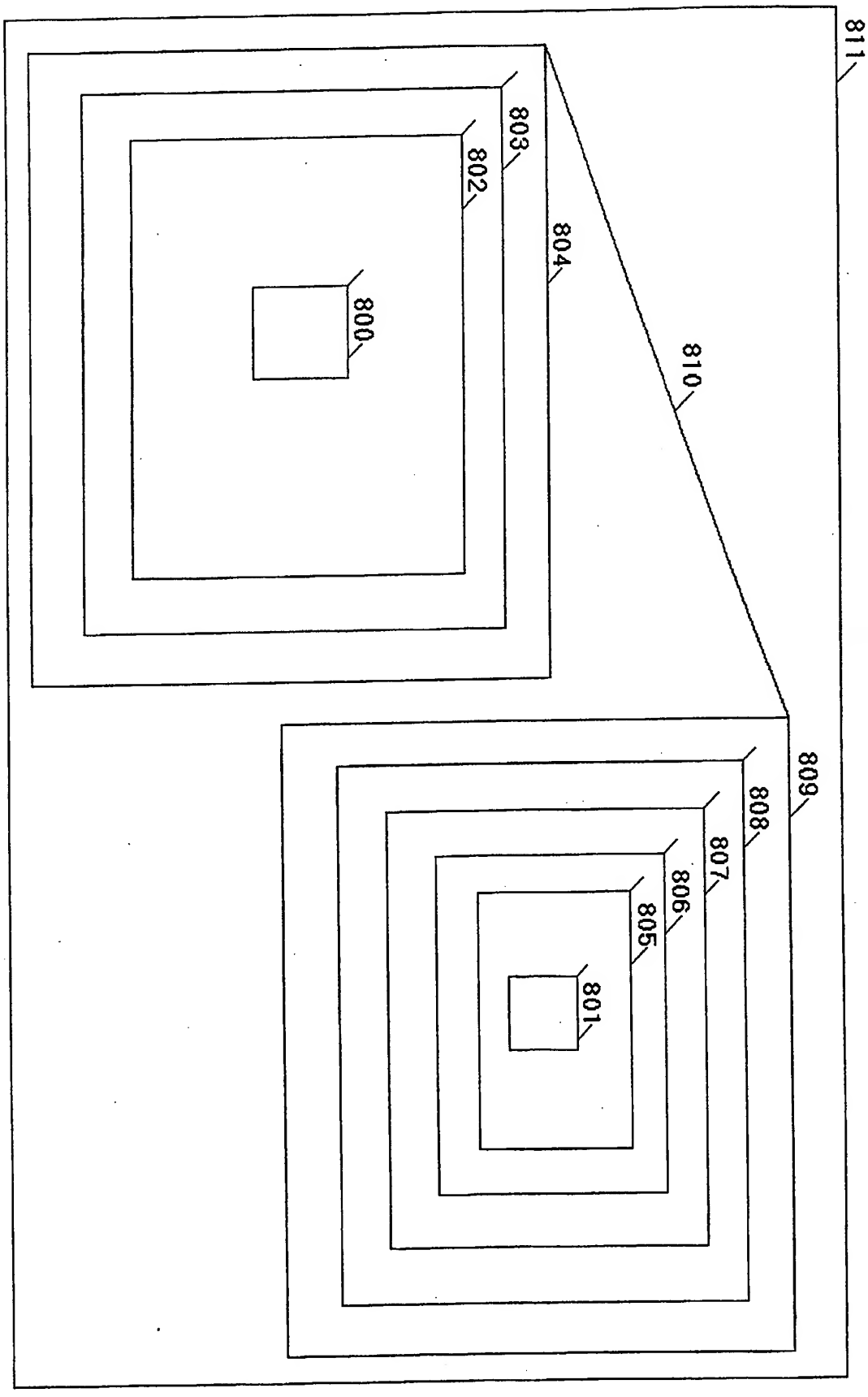


Fig. 31

2387754

ELECTRONIC FILE DISPLAY

The present invention relates to the electronic display of digitally stored files, and in particular to the simultaneous display of a plurality of discrete items of digitally stored information. The invention further relates to the display of hierarchical folders (or directories) and files on a computer display.

It is common for individuals and organisations to store information on digital storage media. So that the information can be accessed by a user, it must be displayed electronically in a human readable manner. Stored information is commonly displayed by apparatus that includes means of differentiating between individual items of information, allowing the selection of any single item. The content of a selected item can then be displayed by the apparatus. Such apparatus ranges from personal computer systems to advertising or presentation apparatus purpose built to display one or more digitally stored visual images.

It is common, particularly with personal computer systems, to store digital information as a list of items, known as files, each differentiable from the others by a unique name. The apparatus can display the list of names on request, from which a particular file may be selected by reference to its name.

Where the number of files is large, a single list becomes impractical for visual interaction by the user. In order to get round this problem, files are arranged into folders or directories, each folder containing a list of files. Each folder may also contain other folders, so that a hierarchical tree structure can be created. Personal computer systems have almost universally adopted this hierarchical tree system as the primary system by which users may store and retrieve data.

Generally speaking, a first folder which contains a second folder among its list of files and folders is said to be the 'parent' of the second folder, and the second folder is said to be the 'child' or 'sub-folder' of the first folder. A first folder which contains a second

folder within the part of the file system it represents is said to be an 'ancestor' of the second folder, and the second folder is said to be a 'descendant' of the first folder. At the highest level there will be a single folder which ultimately contains all the files and folders of the hierarchical tree, and this is known as the 'root folder'.

Although the technical field of the invention is not limited to personal computer systems, for the sake of brevity the terminology of personal computer systems as commonly used and described above will be used throughout this document.

Where a hierarchical tree system is used it is common for apparatus to display on request a list of the files and folders within any one folder, the root folder being displayed in the first instance and display of other folders being requested by selection from a display representing the folders containing them, thus providing a means of navigating the hierarchical tree assisted by its hierarchical categories to gain access to any file or folder within it.

A limitation of such apparatus is that the files and folders within any single displayed list are only distinguishable from each other by the name they have been given, that is there is no indication of what each contains. In general such apparatus may indicate the contents of a file or folder once it has been selected, but recognition of content can play no part in making that selection. This limitation can lead to a number of problems.

A user may have difficulty giving unique and memorable names to all the files that he wishes to store, or cannot associate anything meaningful with files named by somebody else. Files may contain images or documents whose quality is difficult to express with a filename; under these circumstances the user has to resort to opening or selecting files one by one to find out what is inside them.

The same problem of recognition of filenames applies to folders, since a folder may contain further files and sub-folders whose existence is not indicated in any way until that folder has been opened or selected. This leads many users to avoid the use of hierarchical arrangements of sub-folders because once a file is stored in a sub-folder

there is no further visual reminder of its existence until that sub-folder is selected or opened, and the user may forget that it exists.

A further limitation of such apparatus is that the display of a list provides no visual context in which to represent associative relational or referential links between files. Typical examples of such links are hypertext links between hypertext mark-up language (HTML) files, which are now commonly found within hierarchically organised information.

While much useful functionality has been developed and has seen widespread use within the above limitations, the limitations are now being more intensely felt with a number of developments in the use of computer systems. Ever greater numbers of files are now stored in hierarchies of ever greater complexity. Access is required to information structures created by others and delivered by removable storage media or across the Internet. Much use is made of visual imagery within files. The widespread adoption for internet use of HTML and other hyperlinked files has led to an enormous explosion in their use.

There is therefore a need for a system for displaying the contents of a large number of files within a single display area so that the contents of each file can be identified. This can be done if the contents of each file is represented as an image, each image being large enough to view, as long as all the images representing files are visible or can be made to be visible.

Many operating systems use arrangements of 'windows' to display images and file contents. Such systems generally use one or both of two systems for displaying multiple images on a single display area. These are 'tiling', in which images are placed side by side without overlapping, and 'overlapping' in which images are displayed overlapping each other, the sequence of overlapping being recorded and the display being maintained in accordance with that sequence, any image being brought to the top of the sequence by the action of a pointing device (such as a mouse cursor) over any portion of that image that is displayed.

Tiling has the limitation that to fit a large number of images into the display area, the size of each image will have to be very small. Conversely, large images would permit very few to be held within the display area, this limitation being further aggravated by the fact that it is not possible to resize individual images without violating the tiled arrangement of the display area.

When overlapping is used, the small portions of partially obscured images which are visible often do not show enough information for visible recognition. In addition, once an image has been completely overlapped by other images it is unlikely to be seen again and it is difficult to avoid this with more than four or five windows. Furthermore, the user has to spend time positioning and sizing windows to achieve a usable arrangement. The dynamic response at any one time is limited to the manner of displaying a single window. For these reasons existing overlapping mechanisms lose their ease of use if the number of images is too large, and are therefore unsuitable as a method to display a many images. As set against this, overlapping windows do demonstrate by their widespread use how successfully a dynamic response to user interaction can effectively compensate for the obscuring of images caused by overlapping.

Some windows operating systems also use scrolling thumbnail views that allow the contents of files (but not folders) to be seen before they are selected, once they have been scrolled into view.

There are occasions, for example when designing an Internet website, when it is useful to be able to identify links (for example hyperlinks) between documents. It would be useful to be able to display a large number of documents and all the links between them at the same time. However, this is not possible using many existing file management systems without visual confusion resulting.

The representation of files as nodes and links between them as lines is well established. Nodes in 2D space only give a limited environment for expressing links between them before visual confusion results, but positioning nodes in a virtual 3D Space largely overcomes this problem. Systems of positioning nodes on a hyperbolic surface, as

described for example in EP0702330, give very good results for navigating links away from a specific focus. However none of these represent links within a display that also represents the hierarchy in which the information is organised.

This problem is exacerbated if files having links between them are located in different folders. There is therefore a need for displaying a hierarchy of items in such a way that links between items that do not reside in the same folder can be seen or followed.

It may be possible to develop the Cone Tree system as developed by the Xerox PARC institute at Palo Alto USA. and described in US5295243 to express links between items residing in different parts of the hierarchy. The Cone Tree has some capacity to simultaneously represent items in different parts of the hierarchy between which lines can be drawn.

In accordance with a first aspect of the present invention there is provided a method of displaying digitally stored information, comprising creating a virtual 3D space which is rotatable in response to user interaction, representing items of the stored information as labelled nodes within the virtual 3D space, and displaying the virtual 3D space to the user. The virtual 3D space is preferably a Euclidean space.

Each item of information is preferably represented as an image associated with a corresponding node, and which moves with it as the virtual 3D space is rotated.

This allows as many items as the user can reasonably compare to be presented on a single display in an automatically ordered and coherent manner. An image of each item can be displayed at an acceptable size for the user to discern useful aspects of its contents.

Images can be large enough to see the detail within them and as result there is a degree of overlapping of images. In order that the image nearest to the user in the virtual 3D space is not obscured by any of the other images, a first image preferably obscures parts of other images with which it overlaps when the node of the first image is closer to the

user than the nodes of the other images in the virtual 3D space. In other words, each image is preferably displayed so that it always appears as a 2D image facing the user, so that the user receives the impression of a plurality of 2D images suspended in 3D space. The obscuring of information caused by overlapping of images, even if quite considerable, is offset by the intuitive way in which the manner in which they overlap each other can be controlled by rotating the virtual space.

Thus a dynamic response to user interaction is provided which allows the user powerful, intuitive and rapid control over the manner in which images obscure each other whilst enhancing the user's grasp of the ordered and coherent manner in which they are displayed.

The method described above is effective both for the simultaneous display of a small number of images, each image being displayed large enough to view fully, and for the simultaneous display of a large number of images, each image being displayed just large enough to discern its contents, allowing the user to locate a chosen item to be displayed fully by some other means.

The virtual space is preferably rotatable about a single axis, which may be tiltable towards or away from the user in response to instructions from the user. This provides a particularly straightforward and intuitive method for the user to interact with the virtual space to bring required images to the front. It will be understood that other methods for rotating the virtual space may be possible, as long as a simple interface for the user is provided.

Rotation of the virtual space is preferably controllable by the user activating a pointing device, such as a mouse cursor, over an unoccupied part of the virtual space and moving the pointing device. The user can preferably interact with any image using a pointing device to cause that image to be temporarily displayed in a separate window, allow it to be resized, or request appropriate installed application software to display an interface to its associated item of information either within the display area provided for the image or in a separate Window provided by the host operating system.

The items of digitally stored information may include files arranged in folders, and the method preferably further comprises creating a further virtual 3D space representing the contents of a folder and displaying the further virtual 3D space to the user as an image attached to the node corresponding to that folder. The further virtual 3D space should be scaled to fit the area available to it as an image in the virtual 3D space, and it is therefore possible to display hierarchically organised files and folders as a visual hierarchy of nested virtual 3D spaces – i.e. the display comprises a virtual 3D space with further virtual 3D spaces embedded within it, some of which may display yet further virtual 3D spaces embedded within them.

The user can preferably interact with each further virtual 3D space in the same way as with the first virtual 3D space. In other words, he can rotate each further virtual 3D space independently of the first virtual 3D space, and interact with the images contained within it so as to select them for display in a separate window, cause them to be resized, execute application software, etc.

Thus the content of folders as well as their titles can be discerned by the user, which applied recursively results in an unprecedented impression of the entire hierarchical tree system which simultaneously provides a view to several levels of nest depth down every branch of the tree.

This is helpful for gaining an overall impression of the information within a hierarchical tree system, and for providing a visual indication of information density within branches of the tree. Visual cues can be provided for access to members of that hierarchical tree, glimpsing sought after files or folders, which are embedded in unexpected parts of the hierarchical tree, and glimpsing empty folders and files that contain useless or repeated information.

Furthermore a new visual environment is then provided for representing hierarchical tree systems into which other meaningful visual symbols can be introduced, an example being the representation of associative relational or referential links between items of

information throughout the hierarchical tree. The spatial arrangement of nodes representing items of information within a 3D Information Space allows lines to be drawn between them to represent links. Links between items of information (e.g. hyperlinks between HTML files) may therefore be displayed as lines linking the nodes associated with those items of information.

However, two linked files will not always reside in the same folder and will therefore not be displayed as images in the same virtual 3D space. It will therefore not always be possible to display direct lines between images representing two files.

A visual indication of all links between files, including links between files which are not displayed as images within in the same virtual 3D space may be produced by application of the following to the display of each virtual 3D space: a line may be displayed between an image corresponding to a first file and an image corresponding to a second file if the first and second files are linked; a line may be displayed between an image representing a file and an image representing a folder if the file is linked to one or more files embedded within the hierarchy represented by the folder; a line may be displayed between an image representing a first folder and an image representing a second folder if one or more files embedded within the hierarchy of the first folder are linked to one or more files embedded within the hierarchy of the second folder; an external link indicator may be displayed attached to an image representing a first file if the first file is linked to one or more files located in parts of the hierarchy not contained within the parent folder of the first file; and an external link indicator may be displayed attached to an image representing a folder if a file located within the hierarchy of that folder is linked to one or more files located in parts of the hierarchy not contained within the parent folder of that folder.

Thus a coherent expression of all links that exist within a simple list or a Hierarchical tree can be displayed, being of utility in gaining an overview of the structure of linkage within the hierarchical tree.

Preferably, all external link indicators or lines representing links from a file or folder are highlighted if the image corresponding to that file or folder is selected by the user. Lines between nodes which are part of the link may be highlighted by colour or line style. Any external link indicators that are part of the link may be replaced by a highlighted line drawn to a point at the edge of the virtual 3D space closest to the node that represents the embedded virtual space containing it within its parent virtual space.

This further provision allows the complete path throughout the hierarchy of all links from a chosen item to be visually traced, illustrating the distribution through the hierarchical tree of files to which the chosen file is linked, and providing a further utility of visually cued access. The pattern of associative relational or referential linkage throughout the hierarchical tree is visible, limited by images of three dimensional virtual spaces becoming to small for their contents to be discerned at large nest depths.

In accordance with a second aspect of the present invention there is provided a method of displaying digital information stored in the form of hierarchically arranged folders and files, comprising:

creating a first virtual 3D space for a root folder, each file and folder located within the root folder being associated with a unique point in the virtual 3D space;

displaying the first virtual 3D space to the user, each file or folder being represented by an image located at the associated unique point in the virtual 3D space;

wherein the image representing a folder within the first virtual 3D space is a display of a further virtual 3D space representing the contents of that folder.

The first virtual 3D space and the further virtual 3D spaces should be rotatable in response to instructions from the user.

The present invention also provides a computer storage medium having stored thereon a program arranged to perform any of the methods described above.

In accordance with a third aspect of the present invention there is provided apparatus for displaying a plurality of discrete items of information, comprising:

a digital display device controllable by a control unit;

wherein the control unit is arranged to cause images representing the items of information to be displayed on the display device so as to cause an impression to the user of a plurality of 2D images suspended within a virtual 3D space;

and wherein the control unit is further arranged to manipulate the display of the images so that the virtual 3D space appears to rotate in response to instructions from the user. Each image preferably occupies a unique position in the virtual 3D space.

Each image is preferably drawn on the display device adjacent to a position which represents a 2D projection of its position within the virtual 3D space such that, when a first image is closer to the user in the virtual 3D space than a second image, the first image obscures any part of the second image occupying the same region of the display device. The appearance of rotation of the 3D space to the user may be caused by incremental variations of the angle from which the 2D projection is taken, each incremental variation being followed by re-drawing the images.

In order to speed the display process up, a memory image may be held in random access memory of each image to be displayed within a virtual 3D space, each memory image being in a form which can be quickly transferred to the display device or to another memory location; and each memory image may be formed when the images are first displayed, and during subsequent re-display each memory image is directly transferred to the display of the virtual 3D space.

When an image within a virtual 3D space is required to change its appearance, the memory image corresponding to that image is preferably deleted or marked invalid and the display re-drawn. During the re-draw of the display the deleted or invalid memory image may be re-formed to contain the new image and validated.

An image within the virtual 3D space may itself comprise a further virtual 3D space, enabling the display of a hierarchical structure. If the memory image of an image displayed within the further virtual 3D space is deleted or marked invalid then the image in memory of the image within the virtual 3D space is preferably also deleted or marked

invalid.

The items of information preferably include files and folders, a folder being represented as a discrete virtual 3D space. Wherever there exists a link between first and second files which do not share the same parent folder, an iteration which operates on two files or folders is implemented, operating initially on the first and second file, and during each iteration:

if one file or folder is more deeply embedded in the hierarchy than the other then an indication of external links held against the more deeply embedded file or folder is set and the next iteration is carried out replacing the more deeply embedded file or folder with its parent;

if both files or folders are embedded in the hierarchy equally deeply, but do not share the same parent folder, then an indication of external links held against both files or folders is set and the next iteration is carried out replacing both files or folders with their parents; and

if both files or folders share the same parent folder then an implied link is generated between them by adding a reference to the file or folder deriving from the second file to the list of implied links held by the file or folder deriving from the first file and the iteration terminates.

The control of incremental variations of the angle from which the 2D projection may be provided by commands from a computer input device, and produce a dynamic impression of rotation of said three dimensional virtual space which gives the user real time visual feedback whilst using said computer input to device to control the rotation, and provides an intuitive means of altering the manner in which members of said plurality of images obscure each other.

A command from the computer input device may cause any particular image to be temporarily displayed to obscure all else thereby presenting itself fully to the user. On further rotation of the virtual 3D space this effect may be cancelled so that the images obscure each other in the usual way (with the image closest to the viewer obscuring any others). This utility is provided for the convenience of gaining a complete view of any

particular image without the need to find an orientation of the three dimensional virtual space which would allow a complete view of said particular image.

Images displayed using the method of the invention may be sufficiently large to enable detail that is visible within each image to be clearly seen, even though this may increase the degree to which some images are obscured by others, the disadvantages of this obscuring of images being offset by the provisions for the user to control the rotational orientation of the virtual space and hence the manner in which the images obscure each other.

Each image may be accompanied by display of its title, displayed at a size which depends only on the depth of embedding of the virtual space in which it is displayed, the size of the text being reduced by no more than a discernible amount with each increase in embedded depth, so as to allow several reductions in size before the text becomes unreadable.

The computer input device may be a mouse or other pointing device, and the effect of rotation may be controlled by dragging the mouse or pointing device over an unoccupied portion of said three dimensional virtual space with a button depressed such that a horizontal drag will produce said rotation about a vertical axis and a vertical drag will tilt the vertical axis towards or away from the user

The user may use the mouse or pointing device to choose a particular image within said plurality of images and alter the size of said particular image.

A command from the computer input device applied to an particular image, which may be an activation of the pointing device while positioned over that image, may cause actions specific to the image or the information that the image represents;

for example it may request appropriate installed application software to display an interface to its associated item of information, the interface being presented either within the display area provided, or in a separate Window provided by the host operating system.

Where an image within a first virtual space is a further embedded virtual space, any mouse interaction with that image will preferably be interpreted as mouse interaction with the embedded virtual space that it represents, the display of that image being able to dynamically respond to those interactions, thus allowing the rotational orientation of embedded virtual spaces to be individually controlled by the user, and thus providing said embedded virtual spaces with an independent dynamic response that will provide a utility of visually cued access to items of information within them.

Where an image within a first virtual space is a further embedded virtual space, the action of causing a image within the embedded virtual space to be temporarily displayed in front of all others within the same embedded virtual space may also cause the image representing the embedded virtual space within the first virtual space to be displayed in front of all others within the first virtual space, this being required to maintain visual harmony within the display as whole.

Where an image within a first virtual space is a further embedded virtual space, a computer input device interaction may be provided that will cause the embedded virtual space to be displayed either as the root folder of the display, or in a separate Window provided by the Operating System, thus providing a larger view of that embedded virtual space and allowing deeper exploration of the hierarchy.

Where any number of the items of information are linked to any number of other items of information, any one of the images may be selected by the user and, when selected, all line segments which represent a part of any link from its associated item of information may be highlighted by colour or line style, allowing the complete path throughout the hierarchy of all links from a chosen item of information to be visually traced.

Thus the invention generally provides a mechanism for visually over-viewing, performing visually cued navigation and gaining visual access to chosen parts of systems of flat or hierarchically organised information.

Some preferred embodiments of the invention will now be described by way of example only and with reference to the accompanying drawings, in which:

Figure 1 is a schematic block diagram showing the components of a system for displaying files and folders assembled ready for operation;

Figure 2 illustrates an example of a computer screen displaying files and folders using an application operating in accordance with the present invention;

Figure 3 illustrates an example of the display area of the application of Figure 2 following interaction and manipulation by the user;

Figures 4 to 9 show how links between files not residing in the same folder are visually expressed;

Figure 10 illustrates an example of the Display area of the application which causes operation of an embodiment of the invention after some investigative interaction and manipulation by the user;

Figure 11 illustrates an example of the Display area of the application which causes operation of the embodiment of Figure 10 after some investigative interaction and manipulation by the user and after the user has selected one Item for the purpose of highlighting its links;

Figure 12 is a block diagram generally illustrating data and methods used to enable the operation of a system according the invention;

Figure 13 is a block diagram representing a configuration which does not include visualisation of links between files;

Figure 14 is a block diagram representing a configuration which does not include visualisation of links between files or the hierarchical representation of folders;

Figure 15 is a block diagram representing a configuration similar to that of Figure 14 which operates without use of a file content image repository;

Figure 16 is a block diagram representing a configuration which includes hierarchical representation of folders and the visualisation of links between files but which only produces a static display;

Figure 17 is a schematic block diagram showing an overview of an application in accordance with the invention;

Figure 18 is a schematic block diagram showing an overview of the Core engine of the application of Figure 17;

Figure 19 is a schematic block diagram showing an overview of the differences between the application of Figure 2 and the application of Figure 10;

Figure 20 illustrates, using example user data, the drawing operations involved in rendering a display for the first time.

Figure 21 illustrates, using the same example user data as Figure 20, the drawing operations involved in subsequent rendering of the display while the display does not change its appearance;

Figure 22 illustrates, using the same example user data as Figure 20, the drawing operations involved in rendering the display after rotation of folder;

Figure 23 illustrates, using the same example user data as Figure 20, the drawing operations involved in rendering the display after resizing file;

Figure 24 is a schematic block diagram showing the structure of the Data model of the application of Figure 17 and how the parts of it contain and reference each other;

Figure 25 illustrates a populated data model using example user data;

Figure 26 is a schematic block diagram showing the structure of Visual memory support and how the parts of it contain each other;

Figure 27 is a flow chart showing steps in the operation of the HandleEvent process;

Figure 28 is a flow chart showing steps in the operation of the ItemArrayHandleMouse process;

Figure 29 is a flow chart showing steps in the operation of the ItemGetMouse process;

Figure 30 is a schematic block diagram showing extensions to data structures required for the application of Figure 10; and

Figure 31 illustrates using example user data, the requirements of operation of the DoFarLinks and HighlightFarLinks process.

The following description explains how users of personal computer systems running the Windows 98® operating system can view any selection of folders and HTML, Jpeg, or Gif files residing on the persistent data storage unit and accessible through the Windows 98® operating system. It will be understood that although the description applies to one particular operating system, suitable modifications will allow any operating system to be used.

A system for displaying files is shown in Figure 1 and comprises a personal computer system comprising an Arithmetic and Logic unit 100, Read only memory 101, Random access memory 102, and Input/Output control hardware 103 all connected to an address and data bus 104, the Input/Output control hardware being further connected to a

Display unit 105, a pointing device 106, a keyboard 107, and a system of persistent data storage 108, this computer system hardware being configured or selected for compatibility with the Windows 98® operating system.

The Windows 98® operating system software 109 includes the Windows® graphical user interface 110, the Windows Explorer® application 111 and the Microsoft® Web Browser control 112 installed on the persistent data storage unit so that, in accordance with common practice, the Windows 98® operating system is automatically executed when the machine is switched on;

A collection of HTML, Jpeg, or Gif files 113, which may be of interest to the user and which may therefore require display, reside on the persistent data storage unit and are accessible through the Windows 98® operating system.

A pre prepared executable application file 404 resides on the persistent data storage unit accessible through the Windows 98® operating system, the executable application file being written so that when executed it creates a window which is sensitive to the pointing device and which forms the display area in which the invention operates.

The user initiates operation of the file display system by carrying out the following steps:

1. switch on the personal computer system and wait for the Windows 98® operating system to complete its loading sequence;

2. execute the pre prepared executable application file, which will appear initially as a blank window;

3. execute Windows Explorer® and use it to select any list of files and folders from within any one folder;

4. using the pointing device, drag the chosen selection of folders and files from the Windows Explorer® and drop them into the Display area of the pre prepared executable application window.

Figure 2 illustrates the computer screen shortly after completion of the steps described above, in which the Display window shows the folders and files which have been selected and dragged from the Windows Explorer®. At the stage shown in figure 2 the Display window is ready for user interaction even though only two files within the Display window are showing an image of content. The remaining files will fill out with images at the rate of one or two per second over the following minute or so.

The display window of Figure 2 shows a set of files and folders displayed in two windows, a system explorer window 111 and application window 120. In the system explorer window 111 the files and folders are displayed as a list. In the application window 120 they are represented as 2D images suspended in a virtual 3D space. The folders and files appear to the user to be arranged so that they each have a reference point (e.g. the top left corner) in a planar circle within the virtual 3D space. In other words the images are arranged as if on a carousel, with all the images facing the user. The images representing files are arranged in a virtual circle in the top half of the display window, and the images representing folders are arranged in a virtual circle in the bottom half of the display window.

Each image represents the contents of the file or folder. For example, "Index.htm" is an HTML file including a picture, and the first page of this file is shown as the 2D image. The folder "Classic" contains further files and folders, and so the image representing this folder is itself a virtual 3D space, scaled to the size of the image. Any processes carried out on the virtual 3D space of the display window 121 (described below) can also be carried out on the virtual 3D space of each folder.

Once initialised, the Display window 121 responds to the pointing device as follows:

dragging the pointing device in a horizontal (left, right) movement over an unoccupied part of any three dimensional virtual space with the left button depressed will cause that three dimensional virtual space and its contents to visibly rotate about a vertical axis through its centre;

dragging the pointing device in a vertical (up, down) movement over an unoccupied part of any three dimensional virtual space with the left button depressed will cause the above mentioned axis of rotation for that virtual space to tilt away (drag up) or towards (drag down) the user;

dragging the pointing device in any direction over an unoccupied part of any three dimensional virtual space with the right button depressed will cause that three dimensional virtual space and its contents to visibly scroll or translate as if attached to the pointing device;

a single click of the pointing device while positioned over any of the images within any 3D virtual space will cause that image to be displayed in front of all other images within that 3D space and will also cause any ancestors of the file or folder to be displayed in front of all other images in their own 3D space, this effect being cancelled within any three dimensional virtual space when itself or any of its ancestors are rotated. In other words, a single click of the pointing device over a file located within the 3D space of one of the displayed folders will cause the file to come to the front within the display of the folder, and will also cause the folder to come to the front within the display of its parent 3D space;

dragging the pointing device in any direction over the right or lower edge of any image within any three dimensional virtual space will cause that image to be resized;

a double click of the pointing device while positioned over any image within any three dimensional virtual space will cause that image to fill the display area;

a click on the Back button 122 will cause the parent folder, if there is one, of the file or folder currently filling the display area, to become the folder filling the display area.

Figure 3 illustrates an example of the Display area 121 following some investigative interaction and manipulation by the user.

An enhancement of the system described above provides visualisation of links between items of information and also provides highlighting of the links from a single item of information when it is selected by the user for this purpose by positioning the mouse near the top left corner of the displayed item. The user may keep an item selected for this purpose by clicking over the top left corner of the displayed item so that the mouse may be used for other operations while links from that item remain highlighted and a further click over the top left corner of that displayed item or any other will cancel this effect.

The display of links is shown in figures 4-9.

Figure 4 shows an example of the Display Area showing a first file (700) linked to a second file (701) where both files reside in the same folder. Images representing the files are linked by a solid line.

Figure 5 shows the same linked files as in figure 4, but with the link highlighted.

Figure 6 shows a first file (704) linked to a second file (705) where the first file (704) is a child of a first folder (702) and the second file (705) is a child of a second folder (703) and both the first (702) and second (703) folders reside in the same common folder. Each file (704, 705) has attached thereto a short diagonal line projecting from the top left hand corner of the corresponding image, showing that it is linked to another file in a different folder. The two folders (702, 703) are joined by a solid line.

The line joining the two folders indicates that something within one folder is linked to something within the other folder; as this line does not indicate a direct link between the two folders (702) and (703), but is implied by a link between files (704) and (705) within them, this kind of drawn link will hereafter be referred to as a 'Bridge link'.

The symbol projecting from the top left corners of the two files, (704) and (705), indicate that each one is linked to a file which is not in the same folder as itself nor within any of its descendants, this symbol will hereafter be referred to an 'External link indicator'.

Figure 7 shows the same linked files as in figure 6, but with the link highlighted. The entire path of a single link is now shown in full, with the External link indicators replaced by a solid line to the corner of the image representing the folder.

Figure 8 shows a first file (706) linked to a second file (709) where the second file is a child of a first folder (708), that first folder is a child of a second folder (707) and that second folder resides in the same folder as the first file.

The first file (706)and the second folder (707) are joined by a line indicating that the first file (706) is linked to something within the second folder (707). This is another example of a bridge link.

The first folder (709) has a symbol extending from the top left hand corner, indicating that something within it is linked to a file which is not in the same folder as itself nor within any of its descendants. This symbol is another external link indicator. The second file (709) also has attached thereto an external link indicator.

Figure 9 shows the same linked files as in figure 8, but with the link highlighted. As in Figure 7, the external link indicators have been replaced by solid lines to the corner of the folder so that the entire link from the first file to the second file is shown.

Figure 10 shows an exemplary Display area of the application after some investigative interaction and manipulation by the user, in which links between files are shown as described above.

Figure 11 shows the Display area of Figure 10 after the user has positioned the mouse over the top left corner of a first displayed file (710) in order to highlight its links. It can be seen that the first displayed file (710) is linked to files (711 to 717) and if the view is rotated then the destination of other links from the first displayed file will become visible.

The invention can be put into practice by execution of program code written so as to cause working of the invention when executed on a programmable computer equipped with Random Access Memory and a suitable display device;

alternatively specialised electronic hardware may be constructed equipped with Random Access Memory and a suitable display device to work the invention with greater, lesser or no degree of programmability.

Figure 12 is a block diagram illustrating data and methods which work together to operate the invention. Within Figure 12 the following symbolic conventions are used:

rectangles represent data,

rounded rectangles represent methods,

lines with arrows represent one method applying another,

double headed arrows representing iterative application

and long arrows representing recursive application;

and lines without arrows represent important interactions between methods and data.

The diagram shows three input signals:

'Draw display' 23 which the system illustrated receives whenever the display needs to be drawn,

'User input' 24 which the system illustrated receives whenever the user operates the pointing device,

and 'On idle' 26 which the system illustrated receives whenever there is no

requirement to draw the display nor input from the user.

The diagram also shows an output signal 'request to re-draw display' which will cause the system illustrated to receive a new 'Draw display' signal. If more than one 'request to re-draw display' signal is issued during the handling of any input signal then the system illustrated should only receive one 'Draw display' signal in response.

A root list 1 is held of all files and folders that will be displayed, the list is constructed so that further data may be held against each file or folder, which data may be set and retrieved in association with that file or folder.

Where the root list of files and folders includes folders, then for each of those folders a further separate list 2 is held of the further files and folders within it and a reference 3 to that list is held against the folder;

where these separate lists contain further folders then for each of those further folders a yet further separate list is held in the same way; in this way every file and folder within the hierarchy is listed in a hierarchically structured manner which readily allows the retrieval of the list of files and folders within any one folder.

A 3D co-ordinate 4 is held against each file or folder and a general rule is applied to each of the lists of files and folders to assign values to the 3D co-ordinate held against the file and folders within them, for instance the rule may be that all files and folders within each list are assigned values so that collectively their 3D co-ordinate values form a circular configuration.

A 3D orientation matrix 5 or other means of holding 3D orientation information is held against each virtual 3D space and is used to determine the angle from which the virtual 3D space is projected to the display.

A Z order index table 6 is held against each virtual 3D space and contains references to the files and folders in the list which it displays, these are maintained in a sequence which corresponds to their relative distance from the viewer in terms of the virtual 3D effect.

the sequence of each Z order index table is first calculated before its corresponding virtual 3D space is first drawn and is subsequently re-calculated whenever 3D orientation matrix held against its corresponding virtual 3D space is altered.

A method 7 is provided to draw at a chosen size, a virtual 3D space representing a collection of files and folders wherein an image of each of the files and folders within the virtual 3D space is drawn adjacent to or attached to a node positioned to reflect the 2D projection according to the 3D orientation matrix of the virtual 3D space, of the 3D co-ordinate value held against the file or folder;

and the corresponding Z order index table is used to control the sequence or manner in which file and folder images are drawn so that, in terms of the virtual 3D effect, ones which are closer to the viewer obscure portions of ones farther from the viewer wherever they overlap on the display.

A method 8 is provided to draw the image of a file displayed within a virtual 3D space wherein a representation of the contents of the file is obtained by whatever means are available and scaled to fit the size of the image.

A method 9 is provided to draw the image of a folder displayed within a virtual 3D space wherein the method to draw a virtual 3D space is applied to the collection of files and folders within that folder to draw a further virtual 3D space within the dimensions of the image;

The method of drawing a virtual 3D space further provides that it is constructed so that it can be applied recursively

The angle from which a virtual 3D space is projected to form the display may be varied, by altering its 3D orientation matrix under control of the user or pre-programmed instructions and redrawing the display; whenever the 3D orientation matrix of a virtual 3D space is altered the sequence of its Z order index table of the virtual 3D space is re-calculated.

An impression of continuous rotation may be produced by a sequence of displays, each making incremental changes to the angle from which the virtual 3D space is projected.

A method 10 is provided to respond to actions of a pointing device positioned over a virtual 3D space which may be used to control rotation and lateral position of the virtual 3D space and issue commands specific to visually located files or folders.

The sizes of images of files and folders displayed within a virtual 3D space may be individually or collectively varied under control of the user or pre-programmed instructions - for example the method of responding to actions of a pointing device positioned over a virtual 3D space may respond to a drag of the pointing device initiated over an edge of one of the images within that virtual 3D space and may use the position of the pointing device whilst being dragged to control the size of that image.

Optionally a reference 11 to an 'on top' file or folder, which must be able to hold a null value, may be held against each virtual 3D space, this reference is set to null whenever the virtual 3D space is rotated and may be set by user interaction (preferably any kind of clicking of the pointing device while positioned over the image of the file or folder) or pre-programmed instructions to point at any file or folder within it;

and accordingly the method to draw a virtual 3D space may optionally provide in its sequence or manner of drawing the images within it, that if the reference to an 'on top' file or folder is not null, then the image of file or folder it references will not be obscured by any other images - that is it will appear temporarily on top

The impression of continuous rotation may be synchronised with a user input of a continuous nature such as movement of a pointing device;

for instance the 3D orientation matrix of a virtual 3D space may be calculated as a product of Spin matrix and a Tilt matrix and the method of responding to actions of a pointing device positioned over a virtual 3D space may respond to a drag of the pointing device initiated over the virtual 3D space using left/right movements of the pointing device to apply rotation to the Spin matrix and using up/down movements of the

pointing device to apply rotation to the Tilt matrix; this gives the effect of a virtual 3D space which can be spun about an axis (left/right movements of the pointing device) and the axis can be tilted (up/down movements of the pointing device);

Note: For the impression of rotation to be comfortable and effective and for rotation synchronised with a user input to respond to the user without perceptible delay it is important that the display can be redrawn very quickly, this is achieved by the mechanism described in the next two paragraphs.

A memory bitmap 12 is held against each image of a file or folder displayed within a virtual 3D space holding an exact copy of the image as it is displayed, these are formed and used according to the mechanisms described below,

The method of drawing a virtual 3D space further provides that images of files and folders are drawn to the display of a virtual 3D space in the following manner;

if the memory bitmap held against the file or folder is not formed or does not have the same dimensions as are required for the image then a memory bitmap is formed with the same dimensions as are required for the image and held against the file or folder, the method for drawing the image of the file or of a folder within a virtual 3D space is applied to the memory bitmap to form the image of the file or folder on it and the memory bitmap is then transferred to the display of the virtual 3D space;

otherwise if the memory bitmap held against the file or folder is already formed and has the same dimensions as are required for the image then the memory bitmap is transferred to the display of the virtual 3D space.

Note: The transfer of a stored pre-formed memory bitmap image to the display or to another memory bitmap is in current state of the art a very fast operation and its application in this context avoids:

in the case of a file - the much slower operation of obtaining an image of the contents of the file and scaling it to fit the size of the image

and in the case of a folder - recursive application of the method to draw a virtual 3D space;

once the display and all memory bitmaps associated with it have been formed, subsequent re-displays which do not alter the size of images within it, for example rotation or lateral translation of a virtual 3D space, can be completed very quickly

Note: When a virtual 3D space is re-drawn following the resizing of itself, all memory bitmaps held against images of files and folders within it are detected as being not of the required size and are accordingly re-formed at the new required sizes by the action of the method of drawing a virtual 3D space.

Note: When a virtual 3D space is re-drawn following the resizing of an image of a file or folder displayed within it, only the memory bitmap held against the image of the file or folder resized is detected as being not of the required size and is re-formed at the new required size.

The method of responding to a pointing device positioned over a virtual 3D space further provides that its response to the action of the pointing device located over a further virtual 3D space displayed within it, is to apply the same method of responding to the further virtual 3D space according to the location of the pointing device within that further virtual 3D space;

accordingly the method provides that it is constructed so that it can be applied recursively.

Note: When a deeply embedded file or folder changes its appearance (for example a folder is rotated or a file is resized) then it will need to be re-drawn, but as it is displayed within its parent folder then that parent folder will need to be re-drawn and if that parent itself has a further parent then the further parent will need to be redrawn. In

general it can be said that when a file or folder changes its appearance then itself and all of its ancestors need to be re-drawn, this is achieved by the mechanism described in the next three paragraphs.

A method 13 of bitmap invalidation is provided which for any file or folder deletes the memory bitmap held against it or flags it as invalid and then applies itself recursively to its parent folder, if it has one, to ensure that the memory bitmaps of all of its ancestors are deleted or flagged as invalid.

The method of responding to a pointing device positioned over a virtual 3D space further provides that wherever its response causes a change in the appearance of the image of a file or folder then the method of bitmap invalidation is applied to that file or folder and the display is re-drawn.

The method of drawing a virtual 3D space further provides that where the memory bitmap held against an image is non-existent or has been flagged invalid then that memory bitmap is formed or re-formed applying the methods for drawing the image of a file or folder within a virtual 3D space and any invalid flag is unset before transferring the memory bitmap to the display of the virtual 3D space.

Note: Although the invalidation and re-drawing mechanism of the above three paragraphs disrupts the performance advantage of using the transfer of stored pre-formed bitmap images, it does so in a managed and minimal way. Although many ancestor folders may need to be re-drawn, each one only has to apply the method for drawing the image of a folder within a virtual 3D space to the single child which represents the branch of the tree which contains the file or folder that caused the invalidation, all other children of those ancestors are drawn by transfer of their stored pre-formed bitmap images

Optionally a method 14 may be provided to set a file or folder 'On top' which sets the reference to an 'on top' file or folder of the virtual 3D space displaying its parent folder to point at that file or folder and then applies itself recursively to its parent folder, if it

has one, to ensure that all of its ancestors are also set on top; this method is then applied throughout all methods of the invention in place of directly setting the reference to an 'on top' file or folder of a virtual 3D space.

Optionally a method 14 may be provided to unset the 'On top' member of a virtual 3D space which if the 'On top' member is a folder, applies itself recursively to the virtual 3D space displaying the contents of that folder and then sets the reference to an 'on top' file to null; the recursive application ensures that all of its descendants are not holding any of their members 'On top'; this method is then applied throughout all methods of the invention in place of directly clearing the reference to an 'on top' file or folder of a virtual 3D space.

Optionally any folder within the hierarchy held can be made the root virtual 3D space of the display by applying the method of drawing a virtual 3D space to the list of files and folders within it in the first instance when forming the display. Preferably any descendant folder visible within the display can be selected as the root of the display by an action of the pointing device (e.g. double click) whilst positioned over that folder; and the parent folder of the folder which is currently the root of the display can be selected as the root of the display by a similar action of the pointing device whilst not positioned over any file or folder displayed.

Optionally any file within the hierarchy can be fully displayed by applying the method of drawing the image of a file within a virtual 3D space to that file in the first instance when forming the display, or by commanding appropriate installed application software to display that file. Preferably any file visible in the display may be selected for this purpose by an action of the pointing device (e.g. double click) whilst positioned over that file.

A list 15 of linked files is held against each file, the list contains references to other files to which the file is linked, each reference also indicates whether the file referenced is in the same folder as the file holding the list; the references held must be of an immediate type such memory pointer, or offset into an array,

Note: It is helpful to arrange the lists of files and folders that represent the data model into an indexable array and implement the lists themselves as indexable arrays, any file can then be uniquely referenced, using the index to the list containing it and its index within that list.

A method 16 is provided to populate the list of linked files that is held against any file using whatever indication of links are found within that file, or according to the design of the user or pre-programmed instructions.

The method of drawing a virtual 3D space further provides that for each file, a line is drawn from that file to each file referenced in its list of linked files provided that the file referenced resides in the same parent folder as the file.

Note: The following description refers to 'implicit links', the purpose of implicit links is to indicate linkage within any virtual 3D space between either a file and the contents of a folder, or the contents of one folder and the contents of another folder, this provides a hierarchical overview of linkage between different branches of the hierarchy.

Note: The following description also refers to 'indications of the existence of external links' these are necessary to provide an indication that a file is linked to others for every scenario possible in a hierarchical context;

the scenarios and their indications are:

the file is linked to another sibling file - a link is drawn from the file to the sibling file,

a file is linked to another file which is contained within a sibling folder - an implied link is drawn from the file to the sibling folder,

a first folder contains one of two linked files and a second sibling folder contains the other linked file - an implied link is drawn from the first folder to the sibling folder,

a file is linked to another file which is not contained within it parent - an indication of the existence of external links is drawn.

A list 17 of implicitly linked files and folders is held against each file or folder, the list contains references to file or folders within the same parent folder, to which that file or folder is implicitly linked

An indication 18 of the existence of external links to other files which are not descendants of its parent, is held against each file or folder

A method 19 is provided to generate implied links and indications of the existence of external links resulting from a link from a first file to a second file where the second file does not reside in the same folder as the first file,

wherein an iteration which operates on two files or folders, begins with the first and second files, and during each iteration:

if one file or folder is more deeply embedded in the hierarchy than the other then the indication of external links held against the more deeply embedded file or folder is set and the next iteration is carried out replacing the more deeply embedded file or folder with its parent;

if both files or folders are embedded in the hierarchy equally deeply, but do not share the same parent folder, then the indication of external links held against both files or folders is set and the next iteration is carried out replacing both files or folders with their parents,

if both files or folders share the same parent folder then an implied link is generated between them by adding a reference to the file or folder deriving from the second file to the list of implied links held by the file or folder deriving from the first file, the iteration then terminates.

Note: Where two linked files reside in the same folder no implied links are generated

The method of populating a list of linked files further provides that where the linked file does not reside within the same folder as the file against which the list is held, then the method of generating implied links is applied operating on the file against which the list is held and the linked file.

The method of drawing a virtual 3D space further provides that for each file or folder, a line is drawn from that file or folder to each file or folder referenced in its list of implicitly linked files.

The method of drawing a virtual 3D space further provides that for each file or folder, if the indication of the existence of external links is set for that file or folder then a visible mark to represent the existence of external links is drawn associated with the file or folder.

A highlight flag may be held against every reference to linked or implicitly linked folders and any indications of the existence of external links, the normal state of these flags is to be unset.

A method 27 is provided to highlight any link within a list of linked files wherein an iteration which operates on two files or folders, begins with the file against which the list is held and the file referenced by the link, and during each iteration:

if it is the first iteration and both files have the same parent folder then the highlight flag held against the link is set and the iteration terminates;

if one file or folder is more deeply embedded in the hierarchy than the other then the highlight flag of the indication of external links held against the more deeply embedded file or folder is set and the next iteration is carried replacing the deeply embedded file or folder with its parent;

if both files or folders are embedded in the hierarchy equally deeply, but do not share the same parent folder, then the highlight flag of the indication of external links held against both files or folders is set and the next iteration is carried out replacing both files or folders with their parents;

if both files or folders share the same parent folder and it is not the first iteration, then the implied link between them is found by searching the list of implied links held by the file or folder deriving from the first file for a reference to the file or folder deriving from the second file, when the implied link is found its highlight flag is set and the iteration terminates.

The method of highlighting any link within a list of linked files further provides that it can
remove the highlight from any link within a list of linked files by substituting any setting of highlight flags within its method with unsetting of the same flags.

The method of drawing a virtual 3D space further provides that wherever the highlight flag held against a link or implied links is set, then the line drawn representing that link or implied link is drawn with a distinct color or line style to represent a highlighted state.

The method of drawing a virtual 3D space further provides that wherever the highlight flag held against an indication of the existence of external links is set, then a line is drawn from the file or folder against which the indication of the existence of external links is held, to a point at the edge of the display of that virtual 3D space, using the same distinct colour or line style used for highlighted links and implied links.;

the point to which the line is drawn should be preferably as close as possible to the general point of attachment of the display of an embedded virtual 3D space to its position within its it parent virtual 3D space, so that if the virtual 3D space has a parent then there will be a visual continuity between the line drawn and the associated highlighted implied links that will be visible within its parent virtual 3D space emanating from its position within that parent virtual 3D space

The method of responding to a pointing device positioned over a virtual 3D space further provides that an action associated with the pointing device while positioned over a chosen file may cause:

the method of highlighting any link within a list of linked files to be applied to each link within its list of linked files, the method of bitmap invalidation to be applied each of the linked files and the chosen file, and the display to be redrawn, the method of highlighting any link within a list of linked files may set or unset highlight flags according to the action associated with the pointing device

Note: When the display of a virtual 3D space is first formed and whenever it is resized the method of drawing the image of a file within a virtual 3D space must be applied for every file visible within it and every application of the method involves obtaining a representation of the contents of a file; in the case of files stored in magnetic disk or files whose visualisation is complex, this can be time consuming and cause unacceptable delays in drawing the display. The following paragraphs describe a convenient contingency for these circumstances.

A file content image repository 20 is provided whose purpose is to hold an image representing the contents of each file stored in such a way that it can be readily retrieved and transferred to a memory bitmap;

and a reference 21 to a file content image within the file content image repository is held against each file; this reference is initially set to a null value to indicate that no file content image is available.

Note: A simple implementation of the file content image repository would be an indexable array of memory bitmaps, the index of a file content image within the array could then be used as the reference to a file content image held against the associated file. It may be preferable that the memory bitmaps within the repository are of a size which is similar to that at which files are likely to be displayed within a virtual 3D space and the images of content obtained are reduced to fit them.

A method 22 is provided to load an image of file content into the file content image repository for a given file,

wherein an image of file content is obtained by whatever means are available, is put into the file content image repository and the reference to a file content image held against the file is set to reference the new file content image.

The method of drawing the image of a file within a virtual 3D space further provides that if the reference to a file content image held against the file is not a null value, then the method operates by extraction and transfer of a file content image referenced from the file content image repository using the reference to a file content image.

The method of drawing the image of a file within a virtual 3D space preferably provides that if the reference to a file content image held against the file is a null value, then a simple indication of non-availability of image of file content is displayed and no action is taken to form an image of file content..

Note: The use of a simple indication of non-availability of image of file content when a file content image is not yet available allows the display to be formed and respond to user interaction immediately, albeit without any images of file content.

A preferred way of applying the method of loading an image of file content to populate the file content image repository is to arrange that the method of loading an image of file content is applied to each file in turn whenever the system is not busy responding to user interaction;

and each time the method of loading an image of file content completes for a given file the method of bitmap invalidation is applied to the file and the display is re-drawn.

Optionally where the operating environment allows simultaneous execution of multiple execution threads, the method of loading an image of file content into the file content image repository may start whatever process is used to obtain an image of the content of the file in a separate execution thread and return immediately allowing the system to

continue responding to user interaction;

in combination with this an event may be provided which on completion of the associated separate execution thread puts the image obtained into the file content image repository, sets the reference to a file content image held against the file to reference that image, applies the method of bitmap invalidation to the file and causes the display to be re-drawn.

Note: The result of the arrangement described above is that the display is immediately available and responsive to user interaction and images of file content gradually appear during an initial period until they are all present.

Figure 12 represents the presence in the invention of all functionality described above; Figures 13 - 16 represent alternative fully working configurations but with less functionality:

Figure 13 represents a configuration which does not include visualisation of links between files - this would be appropriate where there are no links between the files displayed or the links are of no interest to the user.

Figure 14 represents a configuration which does not include visualisation of links between files or the hierarchical representation of folders - this configuration may be chosen for its simplicity of construction where the files and folders to be displayed are not arranged in a hierarchy, that is there are no folders, or there is for some reason only a requirement to display the first or root level of the hierarchy.

Figure 15 represents the same configuration as figure 14 but operates without use of a file content image repository - the file content image repository is not needed if the image of the content of a file can be obtained from the native stored form of the file without significant delay; this would be the case for instance if the native stored form of the file is as a bitmap image held in random access memory.

Figure 16 represents a configuration which includes hierarchical representation of

folders and the visualisation of links between files but only produces a static display.

Other configurations are possible.

The invention may operate without its provisions relating to calculating 2D projections of 3D points and calculating the Z order of displayed images wherever hardware or software environments in which it operates provide these operations as a service, in which case the invention operates using these services.

When implemented in an environment in which a 3D engine is available to present a 3D model to the display it may be advantageous that nodes and lines within a virtual 3D space are defined within the 3D model and projected to the display by the 3D engine, however it may be preferable that the images displayed within the virtual 3D space are not represented in the 3D model as thin flat 3D objects, but rather as 2D images to be displayed always facing the user, attached to their respective nodes and overlapping each other according to the Z order of their respect nodes .

When implemented in conjunction with a true 3D display such as might be achieved by a digitally controlled dynamic holograph then the images displayed within the virtual 3D space will necessarily have to be shown as thin flat 3D objects and the orientation of these objects will have to be adjusted as the virtual 3D space rotates to ensure that they are always facing the user.

A detailed method for putting the invention into effect will now be described. A personal computer system complete with all of its hardware (Arithmetic and Logic unit (100), Read only memory (101), Random access memory (102), Input /Output control hardware (103), address and data bus (104), Display unit (105), pointing device (106), keyboard (107), and system of persistent data storage (108)) and the Windows 98 operating system (109) (Windows GUI (110), Windows Explorer (111) and Web Browser control (112)) is generally purchased as a single unit of existing state of the art technology. Only the pre prepared executable application file (404) is prepared

specifically to cause operation of the invention, therefore the remainder of this discussion will focus on preparation of the pre prepared executable application file.

The pre prepared executable program file (404) is produced from source code written in a structured programming language, this source code being called by and making calls to the application programming interface (API) of the operating system. The source code is compiled into executable program code and calls to and from the API are resolved by a compiler and linker. The result of the compilation is a named executable program file stored within the file system. The invention is then performed by commanding the operating system to execute the same named executable program file whose effect when executed is hereafter referred to as 'the Application'.

The following description specifies how the aforementioned source code may be written by describing configurations and actions within the computer system which the executable program code it produces should cause when executed, it being normal and routine practice in the art to produce working source code from such a specification.

Figure 17 is a schematic block diagram showing an overview of the application which causes operation of an exemplary embodiment of the invention. The application comprises:

a visible pointing device sensitive application window (120) provided with an event handler HandleWinProc (123) which receives all message from the operating system destined for the Display window (121) in accordance with common practice and a Back button (122),

a Core engine (405) for putting into effect the manipulation of the virtual 3D space,

a set of File Structure services (124) which are called as required by the Core engine,

a set of File Content services (125) which are called as required by the Core engine,

a definition and means of construction of a Bitmap class (126) which implements a drawing surface representing either the Display area or a bitmap image created in memory and implements a set of drawing and bit transfer operations on that drawing surface,

a SendEvent process (127) which receives events from HandleWinProc event handler (123) and processes them to be sent to the HandleEvent process (416) of the Core engine (405) along with a Bitmap (126) representing the Display area (121), on which Bitmap the Core engine can draw to produce output on the Display area of the Display device,

a definition and means of construction of a Web Browser control class (128) which is able to read any HTML, Jpeg, or Gif file and transfer an image of its content onto a Bitmap (126).

The File structure services (124) implement Windows API calls to return information about file and folders and comprise:

(143) bool IsDirectory(Full path and filename of file or folder) which returns true if the address passed is a folder and otherwise returns false;

(144) void FillListWithDirectoryItems(Pointer to String List, Full path and filename of folder) which fills String List with the members of the folder passed;

(145) bool CanDisplayFile(Full path and filename of file) which returns true if the address passed is a folder or HTML, Jpeg, or Gif file and otherwise returns false;

(146) String GetFileTitle(Full path and filename of file or folder) which returns the title of the file or folder passed.

The File content services (125) create and control instances of the Web Browser control class (128) to display or obtain a Bitmap (126) image of the content of HTML, Jpeg, or Gif files and comprise:

(147) ShowFile(Full path and filename of file) which hides the DisplayWindow, creates a visible Web Browser control sized to exactly cover the Display area and navigates it to the address passed;

(148) CloseFileShowing() which detects if a Web Browser control is visible and if so, destroys it and shows the DisplayWindow;

(149) GetFileImage(Pointer, Full path and filename of file) which creates an invisible Web Browser control, sets its Pointer value member (150) equal to the Pointer passed, initiates navigation to the address passed and returns - the invisible Web Browser control then performs the navigation in its own thread calling SendMessage with GOTFILEIMAGE as described below when the navigation is complete and the control holds an image of the file, after which the invisible Web Browser control destroys itself.

The Core engine (405) will now be described in more detail.

Figure 18 is a schematic block diagram showing an overview of the Core engine. The diagram and its components are in many ways analogous to Figure 12 and its components, the core engine being a specific implementation of the workings earlier described in general terms with reference to Figure 12.

The Core engine comprises the following operational units:

a Data Model unit (406) to contain in memory the information that is to be displayed,

a Data Populator unit (160) to populate the Data Model (406) with files and folders selected by the user,

a Display Unit (161) to paint within the Display Area

and a Control Unit (162) to manage user interaction and dynamic response.

The Data Model Unit (406) contains:

the Array of ItemArrays (307) which holds information about the plurality of files and folders being displayed and its hierarchical structure, each ItemArray (311) representing a folder and each Item (322) within that ItemArray representing a file or folder within that folder;

and the ReferenceImageArray (306) which holds images of the visual content of the files to be displayed and provides the display unit with a means of repeatedly accessing images of the visual content of the files without the delay incurred by opening them each time they are needed for a paint operation or the system problems caused by a proliferation of open files in memory.

The Data Populator unit(160) consists of the following process:

the Populate process(407), which takes a list of files and folders passed to it by the HandleEvent(416) process when it receives a WM_DROPFILES message, and from that list, populates the Array of Item Arrays (307), calling itself recursively as it does so;

the Arrange process(408) which generates a three dimensional geometry with which to display the information, the Arrange process is called following the Populate process ;

the GetNextFileImage process(426) which calls the GetFileImage (147) process of the File content services (125) to get an image of the content of a file, the GetNextFileImage process is called by the HandleEvent (416) process when it receives an IDLE message;

and the OnGotFileImage process(425) which is called by the HandleEvent (416) process when it receives a GOTFILEIMAGE message, and stores the image passed in the ReferenceImageArray (306)

The Display unit (161) provides a Visual Memory Support structure (409) which ensures that information required for drawing the display can be accessed as quickly as possible and comprises:

the Array of Z Arrays (344), each Z Array (353) holding information about the screen position and Z order of the images representing files and folders within a 3D virtual space, provided to avoid repeated recalculation of geometrical transformations and Z sorting when the orientation of the 3D virtual space has not changed, the GetZArray (411), GetZElement (419), and Zsort (412) processes being provided to access and maintain the Array of Z Arrays;

and the VisualItemsArray (345), each VisualItem (362) holding a memory Bitmap (126) of a file or folder exactly as it is displayed within the 3D virtual space to which it belongs, provided to avoid the repetition of either:

stretching the associated Reference Image to the required size in the case of a displaying a file image,

or reconstruction of a 3D virtual space in the case of a displaying a folder image,

the GetVisualItem(414) and Invalidate (415) processes being provided to access and maintain the VisualItemsArray (345).

Painting of the Display area (121) is initiated by a call from the HandleEvents (416) process to the DrawItemArray process (410) passing a Bitmap (126) representing a bitmap image stored in memory on which to paint, and is carried out by iterative and recursive interaction between the DrawItemArray process (410) and the DrawItem process (413) as follows:

the DrawItemArray process (410) calls the GetZArray process (411),

which GetZArray process either:

returns an existing ZArray (353) if it is marked valid (the fast route),

recalculates the values of an existing ZArray and returns it if it is marked invalid,

or creates a new ZArray if one doesn't exist, calculates the values and returns it;

the DrawItemArray process (410) uses the ZArray returned, to sequence the order in which its constituent items are painted to the Bitmap passed and in what position each item is painted, painting each item by a call to the DrawItem process (413) passing the same passed Bitmap and the Point the item is to be drawn at;

the DrawItem process (413) calls the GetVisualItem (414) process to gets a Visual Item bitmap which bitmap:

if it is marked valid is transferred directly to the Bitmap passed at the Point passed (the fast route),

otherwise:

if the Item represented is a file then the a Reference Image Bitmap is obtained from the ReferenceImageArray (306) which is stretched to fit the Visual Item bitmap which Visual Item bitmap is then marked valid and transferred to the Bitmap passed at the Point passed,

or if it is a folder then a recursive call is made to the DrawItemArray process (410) passing the VisualItem bitmap which Visual Item bitmap is then marked valid and transferred to the Bitmap passed at the Point passed.

The interaction of these two processes builds a layered structure of bitmaps descending recursively through the hierarchy after which the VisualItemsArray (345) will contain a valid VisualItem Bitmap for every file or folder that is displayed as an image within a 3D virtual space and the ArrayOfZArrays (344) will contain a valid ZArray for every 3D virtual space displayed.

It can be seen from above that:

valid ZArrays allow the fastest execution of the GetZArray process (411) and therefore the fastest execution of the DrawItemArray process (410);

and that valid VisualItems allow the fastest execution of the DrawItem process (413) which fastest execution route avoids further recursive calls to DrawItemArray (410) or stretching of Reference Image Bitmaps;

therefore the fastest repainting of the display will be achieved when all Z Arrays and VisualItems remain valid.

In order to maintain the fastest possible response from the display whilst allowing it to repaint areas whose appearance has changed due to some aspect of dynamic response:

Z Arrays are only marked invalid when the orientation of the 3D Virtual Space they are associated with has been changed since they were last calculated;

and VisualItems are only marked invalid when the representation of the file or folder they represents has changed since they were last updated e.g.:

an image representing them has become available,

the image representing them has resized,

or in the case of VisualItems which represent folders, some aspect of an file or folder displayed within them or the orientation in which they are displayed has changed.

The Invalidate process(415) is provided to ensure that when a VisualItem is marked invalid, any VisualItems within which they are displayed are also marked invalid, it is called when for any reason when due to some aspect of dynamic response an Item has changed its appearance.

Figures 20 - 23 illustrate how invalidation and repainting is controlled to minimise time consuming operations during repainting.

Figure 20 shows a folder (916) which contains two files (900) and (901) and two folders (902) and (903), the folder (902) contains two further folders (904) and (905) each of which contains two further files (908-911), the folder (903) contains two further folders (906) and (907) each of which contains two further files (912-915). Each file or folder is marked with letters which represent the operations required to draw them for the first time where:

DF signifies Draw the Folder,
CZ signifies Create ZArray,
SR signifies Stretch Reference image.

it can be seen from figure 20 that to draw the system represented for the first time requires drawing a folder 7 times and creating a ZArray 7 times and stretching a reference image 10 times.

Figure 21 shows the same system and the files and folders and operations required to redraw it if it is not invalidated due to any change in its appearance where

B signifies Blit or Bit transfer,

it can be seen that only 4 Bit transfers are required.

Figure 22 shows the same system and the files and folders and operations required to redraw it after the three dimensional space representing folder (907) has been rotated where

RZ signifies Re-calculate ZArray;

and Figure 23 shows the same system and the files and folders and operations required to redraw it after the file (909) has been resized;

it can be seen that in both cases much of the redrawing is still achieved by rapid blit or bit transfer operations.

The Control (162) unit is driven by HandleEvents process (416) which receives events from the SendEvent process (127) of the Application.

On receiving a WM_DROPFILES message with a list of files and folders representing a part of the file system selected by the user, the HandleEvents process (416) calls the Populate process (407) to fill the ArrayOfItemArrays (307)with a information representing the portion of file system passed and then calls the Arrange process (408) to generate a geometrical structure for each ItemArray (311).

Once the ArrayOfItemArrays (307) is populated, the HandleEvents process (416) will call the DrawItemArray process (410) whenever it receives a Paint or Size message or if the processing of any other message or event indicates a requirement to repaint the Display area (121).

The HandleEvents process (416) calls the ItemArrayGetMouse process (417) whenever it receives a mouse event from the Display area (121). Propagation and response to mouse events is carried out by recursive interaction between the ItemArrayGetMouse process (417), the ItemGetMouse process(422), and the ItemArrayHandleMouse process (420) as follows:

the ItemArrayGetMouse process (417) calls the HitTest (418) process for each of its constituent Items, which HitTest process returns true if the mouse is positioned over that Item and;

if any call to HitTest returns true then the ItemGetMouse process (422) for that Item is called;

otherwise the ItemArrayHandleMouse process(420) is called;

the ItemGetMouse process(422):

recursively calls the ItemArrayGetMouse process (417) if the Item represents a folder,

and otherwise handles the mouse event itself, which may involve:

resizing the image representing the Item if its border is being dragged,

or displaying the file if the event is a double click;

and the ItemArrayHandleMouse process(420) which handles mouse events which are positioned over an unoccupied region of a 3D Virtual Space and may involve:

rotation of the 3D Virtual Space if the event is a LBUTTONDRAG, by calling the Spin & Tilt process;

altering the position of a 3D Virtual Space within its display area if the event is a RBUTTONDRAG;

or if the event is a double click, causing the Folder it represents to become the root folder displayed and whose 3D Virtual Space occupies the main Display area.

Update of the Display to reflect changes resulting from mouse actions is managed by appropriate calls to the Invalidate process and propagating back a return value of true which causes the HandleEvents process (416) to call the DrawItemArray process (410) before exiting.

It is in the current state of the art a normal user expectation of any graphical interface that if you click on any part of a partially obscured visual object then it will be displayed in front or on top of all others. Accordingly the Select process (423) is called by the ItemGetMouse process (422) each time it receives a Mouse Down message. The Select process (423) attributes that Item as being on top within its parent ItemArray and this status is respected by the DrawItemArray (410) and the ItemArrayGetMouse (417) processes, the Select process (423) also ensures that all ancestors of the Item selected are also attributed as being on top within their parent ItemArrays.

It is in the current state of the art also a normal user expectation of any graphical interface that if you commence a mouse drag within a visual object then that visual object should continue to receive all mouse drag messages, even if the mouse is no longer over the visual object, until the mouse is released; this is known in the art as mouse capture. Accordingly a mouse capture mechanism is operated by the ItemArrayGetMouse (417), ItemArrayHandleMouse (420) and ItemGetMouse (422) processes.

Figure 19 summarises changes which should be made to the embodiment described above to effect the enhancement which displays links. The processes represented in the figure by an ellipse are those which form part of the earlier description which need to be changed to make calls to new processes represented by rounded rectangles which are introduced to effect the enhancement.

The following describes, with reference to figure 19, the changes which should be made to the previously described embodiment to effect the enhancement which displays links: The WebBrowser control (128) is modified so that when it handles the OnDocumentComplete event (151), it calls a new GetLinks process (170) which constructs a StringList containing the location of all files that the page that it holds is linked to and calls the SendEvent process (127) passing a DOITEMLINKS message, the PointerValue member (150) of the WebBrowser control (128) and the StringList constructed by the new GetLinks process (170).

The HandleEvent process (416) is modified so that it calls the DoItemLinks process (30) in response to a DOITEMLINKS message. The DoItemLinks process is an implementation of the method of populating a list of linked files 18 described earlier with reference to figure 12 and accordingly calls the DoFarLink process (31) which is an implementation of the method of generating implied links 19 described earlier with reference to figure 12.

The ItemArrayHandleMouse process (420) and the ItemGetMouse process (422) are modified to control the HighlightItemLinks process (36) which calls the HighlightItemLinks process (36). The HighlightItemLinks process (36) is an implementation of the method of highlighting a link between two files 27 described earlier with reference to figure 12.

The DrawItemArray process (410) is modified to call to the new DrawArrayLinks process (32) which calls the DrawItemLinks process (33) for each Item in the ItemArray. The DrawItemLinks process is an implementation of the extensions to method of drawing a virtual 3D space 7 which effect the display of links described earlier with reference to figure 12.

Extensions to the data model 40 include implementation of the lists of links 15, lists of implied links 17 and external link indicators 18 described earlier with reference to figure 12.

The following is a detailed description of the data structures and processes referred to above. They are first described without implementation of the enhancement which displays links and then the changes which are necessary to effect the enhancement which displays links will be described.

The Bitmap class

The Bitmap class (126) is provided with two constructors:

(129) new Bitmap(Window)- which is used by the SendEvent process to construct an instance of a Bitmap class which represents the Display area (the

Windows Device Context of the Display area is held as the operand of drawing and bit transfer operations),

(130) new Bitmap(Size)- which is used throughout the Core engine to construct an instance of a Bitmap class (126) which creates and represents a bitmap image stored in memory (a new Windows device context is created and a new Windows Bitmap is created of the Size passed and selected into the new Windows device context which is held as the operand of drawing and bit transfer operations);

each instance of the Bitmap class (126)implementing the following operations on the drawing surface it represents:

(131) FillColor(Colour) which fills the entire Bitmap (126) with Colour passed,

(132) BlitTo(Point, Source Bitmap) which transfers the entire Source Bitmap to the Bitmap at the Point passed,

(133) StretchToFit(Source Bitmap) which transfers the entire Source Bitmap to the Bitmap stretching it to fit the Bitmap exactly,

(134) SetPen(Type) which sets the pen for drawing operations to be of the Type passed (this operation is called frequently therefore repeated creation and destruction of Windows Pen objects is avoided by using a static reservoir of pre created Windows Pen objects),

(135) MoveTo(Point) which moves the current pen position to the Point passed,

(136) LineTo(Point) which draws a line on the Bitmap from the current pen position to the Point passed,

(137) Rectangle(Rectangle) which draws an outline of the Rectangle passed onto the Bitmap.

(138) SetFont(Nest Depth) which sets the font for text operations to be appropriate for the Nest Depth passed, a working formula for calculating font height being 'Font height equals fifteen minus three times the Nest Depth' (this operation is called frequently therefore repeated creation and destruction of Windows Font objects is avoided by using a static reservoir of pre created Windows Font objects),

(139) SetTextColor(Colour) which sets the text colour for text operations to be the Colour passed,

(140) SetTextBackgroundColor(Colour) which sets the text background colour for text operations to be the Colour passed,

(141) Size GetTextSize(Text) which returns the size that the Text passed would occupy drawn using the current font,

(142) TextOut(Point, Text) which prints the Text passed onto the bitmap at the Point passed.

The Web Browser control class

The Web Browser control class (128) which is built around the Microsoft Web Browser control, implements the OnDocumentComplete event handler(151) and sets the Pointer value member(150) to NULL on construction. The OnDocumentComplete event handler is called when the navigation is complete and the control holds an image of the file and performs the following operations when called:

if the Pointer value member is NULL then does nothing and returns;

otherwise:

creates a local (destroyed on exit from the process) Bitmap (126) instance using the new Bitmap(Size) constructor (130) passing the size of the Web Browser control,

paints the Web Browser control onto the bitmap instance using the OleDraw Windows API call,

calls the SendEvent process (127) passing the following parameters:

message = GOTFILEIMAGE, x = control width, y = control height,

pVoid1 = Pointer value member, pVoid2 = address of Bitmap,

then posts a message to the Web Browser control to destroy itself.

The SendEvent process

The SendEvent process (127) creates a Bitmap (126) representing the screen and prepares events messages and parameters which are passed in a call to the HandleEvent process (416) of the Core engine.(405).

The SendEvent process is defined according to C++ syntax:

void SendEvent(UINT Message, int x, int y, void* pVoid1, void* pVoid2);

wherein:

'Message' indicates the type of message or event,

'x' & 'y' are used to pass position values where appropriate,

'pVoid1' & 'pVoid2' are used to pass further parameters by reference;

and operation of the core engine is achieved by calling the SendEvent process from the HandleWinProc (123) event handler of the application window as follows:

In response to an Initialise event

SendEvent(INITIALISE, 0, 0, NULL, NULL);

In response to WM_SIZE message

SendEvent(WM_SIZE, LOWORD(lParam), HIWORD(lParam), NULL, NULL);

In response to WM_PAINT message

SendEvent(WM_PAINT, 0, 0, NULL, NULL);

In response to WM_DROPFILES message

SendEvent(WM_DROPFILES, 0, 0, (void*)wParam, NULL);

In response to WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDBLCLK, WM_RBUTTONDOWN and WM_RBUTTONUP messages

SendEvent(message, LOWORD(lParam), HIWORD(lParam), (void*)&wParam, NULL);

In response to an OnIdle event

SendEvent(IDLE, 0, 0, NULL, NULL);

In response to Back button pressed

SendEvent(BACKBUTTON, 0, 0, NULL, NULL);

When called the SendEvent process (127) creates a local (destroyed on exit from the process) Bitmap (126) instance representing the Display area by using the new Bitmap(Window) constructor and performs the following modifications to the message and parameters before passing them with the new Bitmap instance to the HandleEvent process (416) of the Core engine (405):

In response to a WM_DROPFILES message

uses the Windows HDROP passed by reference in pVoid1 to fill a string list with the filenames of the files and folders dropped, and passes a pointer to the string list to the HandleEvent process of the Core engine as the pVoid1 parameter;

In response to a WM_MOUSEMOVE message

use the UINT nFlags passed by reference in pVoid1 to determine if either of the left or right buttons of the pointing device are held down and:

if the left button is held down then calls the HandleEvent process of the Core engine passing LBUTTONDDRAG as the message parameter and the amount by which the pointing device has been dragged as the x and y parameters;

if the right button is held down then calls the HandleEvent process of the Core engine passing RBUTTONDDRAG as the message parameter and the amount by which the pointing device has been dragged as the x and y parameters;

if neither button is pressed then calls the HandleEvent process of the Core engine without altering any parameters;

In response to all other messages and events calls the HandleEvent process of the Core engine without altering any parameters.

Data Model

Figure 24 is a schematic block diagram showing the structure of the data model (406) and how the parts of it contain (300, 301, 302) and reference (303, 304) each other.

The Data Model globals (305) are initialised when the Core engine (405) is created and comprise:

the ReferenceImageArray(306) which is an array of Bitmaps (126) initialised as empty, which will be used to hold reference images of the files represented;

the ArrayOfItemArrays(307) which is an array of ItemArrays (311) initialised as empty, the zero element of which is used to represent the root folder,-and the remaining elements to represent all other folders

g_ArrayIndexLoading(308) and g_ItemIndexLoading(309) which are an integers initialised to NONE and are used by the GetNextFileImage process (426) during IDLE events;

and g_NumLoading (310) which is an integer initialised to zero and used by the GetNextFileImage (426) and OnGotFileImage (425) processes to control the number of files allowed to load asynchronously at the same time.

The Item Array structure (311) represents the format for storing the list of files and folders within a folder, and contains:

an array of Items (312) representing the list of files and folders

a SpinMatrix member (313), which is a 3D transformation matrix initialised to unity, and represents rotation about a vertical axis of the 3D virtual space which represents the folder,

a TiltMatrix member (314), which is a 3D transformation matrix initialised to unity, and represents the tilt of the above mentioned vertical axis towards the user,

a ScrollOffset member (315), which is a Point value (x and y co-ordinates both initialised to zero) and represents the offset from the centre of the display area of the centre of the 3D virtual space when displayed,

a Parent Item Address member (316), which is an ItemAddress (317) referencing its parent Item if it has one, the Item Address-structure (317) consisting of a ParentArrayIndex member (318) and an ItemIndex member(319) both initialised to NONE,

a ZArrayIndex member (320) which is an integer initialised to NONE, and when not equal to NONE is an index into the Arrayof ZArrays (344) of its associated ZArray (353),

and a NestDepth member(321) which is an integer indicating the nested depth of the folder it represents within the hierarchy of files and folder represented.

The Item Structure(322) represents the format for storing information about each file or folder within a folder and contains:

a Filename member(323) which is a text string initialised as empty and is used the store the full path and filename of the file or folder;

a Title member(324) which is a text string initialised as empty and is used the store a short title which will be displayed for the file or folder;

an ImageIndex member(325) which is an index initialised to NONE, and when not equal to NONE is an index into The ReferenceImageArray (306) of a Bitmap (126) holding a visual representation of the contents of the file;

an Item Array Index member(326) which is an index initialised to NONE, and when not equal to NONE is an index into the ArrayofItemArrays (307) of the ItemArray (311) representing the folder;

a ThreeDimensionalCoordinate member (327) which is a 3D Point value (x and y and z co-ordinates all initialised to zero) used to indicates the position of the Item within its three dimensional logical space.

a ImageSizeRatioX member(328) which is a ratio initialised to unity and indicates the width of the image displayed to represent the item, with respect to the default width of images displayed in the same 3D virtual space,

a ImageSizeRatioY member(329) which is a ratio initialised to unity and indicates the height of the image displayed to represent the item, with respect to the default height of images displayed in the same 3D virtual space,

a ParentArrayIndex member(330)-which is an index into the ArrayofItemArrays (307) of the ItemArray (311) to which the Item belongs;

a OwnIndex member(331) which is an index into the ItemArray (311) to which the Item belongs of the Item itself;

a NestDepth member(332) which is an integer indicating the nested depth of the item it represents within the hierarchy of files and folder represented,

and a VisualItemIndex member(333) which is an integer initialised to NONE, and when not equal to NONE is an index into the VisualItemsArray (345) of its associated VisualItem (362).

Figure 25 illustrates the Data model configured for a particular hierarchical system of files and folders illustrated in box (920), and shows the ArrayOfItemArrays (307) containing 4 ItemArrays (311) representing the 4 folders each of which contains a number of Items (322) each labelled by its Title member (324) and either indicating that its ItemArrayIndex member (326) references an ItemArray within the ArrayOfItemArrays (307) or that its ImageIndex member (325) references a reference image bitmap (126) in the ReferenceImageArray (306).

Data model processes

GetParentItem

The GetParentItem process (427) is passed an Item (322) and an ItemArray (311), one of which is always NULL and the other the operand of the process, in both cases it returns the parent Item if there is one.

The GetParentItem process operates as follows:

if the ItemArray passed is Null and the Item passed is the operand then

indexes a first ItemArray from the ArrayOfIItemArrays (307) using the ParentArrayIndex (330) member of the Item passed;

The GetParentItem process then indexes a second ItemArray from the ArrayOfIItemArrays (307) using the ParentArrayIndex (318) member of the ParentItemAddress member (316) of the ItemArray (311) passed or first indexed and indexes an Item (322) from the second ItemArray indexed using the ItemIndex member (319) of the ParentItemAddress member (316) of the ItemArray (311) passed or first indexed. The process then exits returning the Item indexed. If either members of the ParentItemAddress member (316) of the ItemArray (311) passed or first indexed have a value of NONE then the process exits returning NULL.

Data populator processes

Populate

The Populate process (407) is passed

an ItemAddress (317) holding the address within the ArrayOfItemArrays (307) of the calling Item, both members of ItemAddress retaining a value of NONE if the Populate process is called by the HandleEvents process (416);

NestDepth which is an integer value holding the value of the NestDepth member (332) of the calling Item, or a value of zero if the Populate process is called by the HandleEvents process (416);

a List of text strings holding the full paths and filenames of a list of files and folders;

The Populate process creates a new ItemArray (311), sets the NestDepth member (321) of the new ItemArray equal to the NestDepth passed and the ParentItemAddress member (316) equal to ItemAddress passed, adds the new ItemArray to the ArrayOfItemArrays (307) and sets the ParentArrayIndex member (318) of ItemAddress equal to the index of the new ItemArray within the ArrayOfItemArrays;

it then iterates through the List and for each list member:

{

creates a new Item (322) , adds the new Item to the new ItemArray and sets the OwnIndex member (331) of the new Item equal to the index of the new Item within the new ItemArray, sets the NestDepth member (332) equal to NestDepth, the Filename member (323) equal to the list member, the ParentIArrayIndex member (330) to the ParentArrayIndex member (318) of ItemAddress, the Title member (324) equal to the value returned from a call to the GetFileTitle process (146) passing ListMember, sets the ItemIndex member (319) of ItemAddress equal to the OwnIndexMember (331) of the new Item and then calls the IsDirectory process (143) passing ListMember;

and if the IsDirectory process returns true then

{

calls the FillListWithDirectoryItems process (144) passing ListMember, and calls the Populate process (407) recursively passing ItemAddress, NestDepth +1 and the string list obtained from FillListWithDirectoryItems;

}

}

A single call to the Populate process (407) by the HandleEvents process (416) causes the ArrayOfItemArrays (307) to represent the entire branch of the file system passed, the zero element representing the root folder, the remaining elements representing all other folders.

Arrange

The Arrange process (408) operates on the ArrayOfItemArrays (307) iterating through the ItemArrays within and for each ItemArray:

assigns values to the ThreeDimemsionalCoordinate member (327) of each of the Items within, so that collectively they form a coherent spatial configuration, and sets the TiltMatrix member (314) of the ItemArray to a slight tilt (visually more revealing)

GetNextFileImage

The GetNextFileImage process (426) is called by the HandleEvent process (416) each time it receives an IDLE event. The purpose of the GetNextFileImage process (426) is to initiate obtaining an image from the contents of a single file each time there is an IDLE event until all images of content have been obtained. Because reading the content of files is time consuming, it is carried out after the display is activated and images of content for each item appear on the display as they are obtained.

The GetNextFileImage process (426) makes use of the g_ArrayIndexLoading (308) and g_ItemIndexLoading (309) global variables to keep track of its position as it iterates

through every Item of every ItemArray in the ArrayOfItemArrays (307), processing one Item during each IDLE event in the following manner:

calls the CanDisplayFile process (145) and if CanDisplayFile returns true then

increments the g_NumLoading (310) global variable and calls the GetFileImage process (147) which initiates obtaining an image for that Item.

The GetNextFileImage process (426) also checks on entry that the value of g_NumLoading (310) has not risen to high and if it is too high then the GetNextFileImage process returns without any further processing.

OnGotFileImage

The OnGotFileImage process (425) is called by the HandleEvent process (416) each time it receives an GOTFILEIMAGE event which happens when an image of the content of a file is ready after a previous call to the GetFileImage process (147).
The OnGotFileImage process (425) is passed:

a reference to an Item which is the Item whose image is ready, a Bitmap holding the image which is ready, and a Size variable holding the Size of the Bitmap.

The OnGotFileImage process (425) decrements the g_NumLoading (310) global variable and then creates a new Bitmap (126) using the Bitmap(Size) constructor (130) passing the size used for reference bitmaps, adds the new Bitmap to the ReferenceImageArray (306) , sets the ImageIndex member (325) of the Item passed equal to the index of the new Bitmap within the ReferenceImageArray, transfers and stretches the Bitmap passed onto the new reference Bitmap by calling the StretchToFit process (133) and invalidates the Item whose image is ready by calling the Invalidate process (415). The next time that Item is Drawn, the DrawItem process (413) will fetch the reference Bitmap referenced by the valid ImageIndex member (325) of that Item.

Visual memory support

Figure 26 is a schematic block diagram showing the structure of the Visual Memory Support (409) and how the parts of it contain (340, 341, 342) each other.

The Visual Memory Support globals (343) are initialised when the Core engine (405) is created and comprise:

the ArrayOfZArrays (344) which is an array of ZArrays, each Zarray (353) holding positional and Z order information about the Items within an ItemArray (311);

the VisualItemsArray (435) which is an array of VisualItems, each VisualItem (362) holding an image of the content of a file or folder exactly as currently displayed;

g_pItemArrayShowing (346)) which is a pointer initialised to NULL and when not equal to NULL, points to the ItemArray (311) which represents the root folder currently displayed;

g_pItemShowing (347) which is a pointer initialised to NULL and when not equal to NULL, points to the Item (322) whose file is currently displayed to fill the Display area (121);

g_ShowingNestDepth (348) which is an integer initialised to zero, and holds the nest depth of the ItemArray (311) which represents the root folder currently displayed with respect to the ItemArray which is the zero element of the ArrayOfItemArrays (307);

g_pWindowMemBitmap (349) which is a pointer initialised to NULL and when not equal to NULL, points to a Bitmap (126) in memory onto which the root

three dimensional virtual space is be drawn, the Bitmap being transferred to the display when complete in one bit block transfer operation;

g_pCapturedItem (350) which is a pointer initialised to NULL and when not equal to NULL, points to the Item (322) whose display is holding the mouse capture during a drag operation of the pointing device;

g_pCapturedItemArray (351) which is a pointer initialised to NULL and when not equal to NULL, points to the ItemArray (311) whose display is holding the mouse capture during a drag operation of the pointing device;

g_bCaptureOn (352) which is a boolean value initialised to false and when true indicates that a mouse drag is currently taking place.

g_WindowSize (366) which is a Size value holding the size of the display area in which the invention operates.

The ZArray structure (353) presents the format for storing information about positional information and Z order of Items within an ItemArray (311) and contains;

an array of Zelements (354) each associated with an Item (322) within the ItemArray;

a bOrientationValid member (355) which is a boolean value initialised to false and when true indicates that the ZArray is valid and the information it holds can be used without recalculation;

a RotationMatrix member (356), which is a 3D transformation matrix initialised to unity, and represents the net rotation of the 3D virtual space, used to avoid repeated recalculation of the product of the Spin and Tilt matrix members of the ItemArray;

a pOnTopElement member (357), which is a pointer initialised to NULL and when not equal to NULL, points to the Item (322) which the user has selected to be presented 'on top' in the display, that is visible in front of all the others.

The ZElement structure (358) represents the format for storing positional and Z order information about each Item within the ItemArray (311) with which it is associated and contains:

an ItemIndex member (359) which is an index initialised to NONE, and when not equal to NONE is an index into the associated ItemArray, of the Item (322) which the ZElement represents;

a NodePosition member (360), which is a Point value (x and y co-ordinates both initialised to zero) which the unscaled position on the screen at which the Item (322) it represents is drawn to the screen;

a Zvalue member (361), which is an integer initialised to zero, and holds the Z position, that is position perpendicular to the display surface, within the virtual space of the Item (322) it represents;

The VisualItem structure (362) represents the format for storing an image of the content of a file or folder exactly as currently displayed and contains:

a bBitmapValid member (363)) which is a boolean value initialised to false and when true indicates that the pMemBitmap member (365) holds an image which is currently valid, and therefore can be used without any further drawing operations being performed on it;

a bBitmapSize member (364) which is a Size value (width and height both initialised to zero) and holds the size of the image;

a pMemBitmap member (365) which is a pointer initialised to NULL and when not equal to NULL, points to a Bitmap which has been created to hold the image;

Display unit processes

DrawItemArray.

The DrawItemArray process (410) is passed an ItemArray (311), a Bitmap (126) and a Size and is called by the HandleEvents process (416) when it receives a WM_PAINT message or the return value from the ItemArrayGetMouse process (417) when handling mouse events is true, it is also called by the DrawItem process (413) when the Item being drawn represents a folder.

The DrawItemArray process (410) calls the FillColor member function (131) of the Bitmap passed to paint a background and then draws the ItemArray passed, onto the Bitmap passed, by calling the GetZArray process (411) to get a ZArray (353) and then iterating through that ZArray and for each ZElement (358) in the ZArray:

{

calculates a position based on NodePosition member (360) of the ZElement, scaled to fit the view area (indicated by the Size passed), offset to place the origin in the centre and further offset according to the value of the ScrollOffset member (315) of the ItemArray passed,

calculates a default display size for Items displayed based on the Size passed,

draws the title of the Item indexed by the ItemIndex member (359) of the ZElement at the position calculated, obtaining the title from the Title member of the Item indexed and using the Bitmap member functions FillColor (131), SetPen (134), MoveTo (135), LineTo (136), Rectangle (137), SetFont (138), SetTextColor (139), SetTextbackgroundColor (140), GetTextSize (141) and

TextOut (142) to draw the text at a suitable size and with a suitable frame around it,

and calls the DrawItem process (413) passing the same calculated position, the same passed Bitmap and the default display size calculated.

}

If the pOnTopElement member (357) of the ZArray has a value other than NULL, then one more iteration is carried out using the ZElement it points to, in order to ensure that it is displayed on top.

A working formula for determining a suitable text size for Item titles is

TextHeight = 15 minus (3 times the value of the NestDepth member (321) of the ItemArray being drawn minus g_ShowingNestDepth (348))

GetZArray.

The GetZArray process (411) is passed an ItemArray (311) and returns a valid ZArray (353) holding information about the screen position and Z order of the images representing files and folders within the 3D virtual space representing the ItemArray passed.

The GetZArray process operates as follows:

if the ZArrayIndex member (320) of the ItemArray passed is not NONE then

{

uses the ZArrayIndex member (320) of the ItemArray to index a ZArray from the ZArraysArray (344) and

{

if the bOrientationValid member (355) of the ZArray obtained has a value of true then

returns the ZArray obtained;

}

}

if the ZArrayIndex member (320) of the ItemArray passed is equal to NONE then

{

creates a new ZArray (353) and adds it to the ZArraysArray (344)setting the ZArrayIndex member (320) of the ItemArray equal to the index of the new ZArray in the ZArraysArray;

}

if the ZArray is empty then

{

fills the ZArray with one ZElement (358) for each Item in the ItemArray passed, setting the ItemIndex member (359) of each new ZElement to the index of its associated Item in the ItemArray passed;.

}

next the GetZArray process calculates the RotationMatrix member (356) of the Zarray by pre-multiplying the SpinMatrix member (313) of the Zarray by the TiltMatrix member (314) of the Zarray and

for each ZElement (358) in the ZArray

{

uses the ItemIndex member (320) of the ZElement to index an Item from the ArrayOfItemArrays (307) and pre-multiplies the ThreeDimemsionalCoordinate member (327) of the Item indexed, by the RotationMatrix member (356) of the Zarray to calculate a ThreeDimemsionalCoordinate value representing the Items position in the virtual space as seen by the user, the NodePostion member (360) of the ZElement is set to the x and y values of the ThreeDimemsionalCoordinate value calculated and the ZValue member (361) of the ZElement is set to the z value of the ThreeDimemsionalCoordinate value calculated;

}

finally the GetZArray process calls the ZSort process (412) passing the ZArray obtained or created sets the bOrientationValid member (355) of the ZArray to true and exits returning the same ZArray.

ZSort

The ZSort process (412) is passed a ZArray (353) and sorts the elements of that ZArray by using a traditional shuttle sort with lower and upper thresholds. The order of sorting being determined by the ZValue members (361) of the Zelements (358) within the ZArray.

DrawItem

The DrawItem process (413) is passed an Item (322) which is to be drawn, a Bitmap (126) to draw on, a point value holding the position on the Bitmap at which the Item is to be drawn, and a Size value indicating the default size for Items drawn onto the Bitmap.

The DrawItem process calculates a Size value whose width is the product of the width of the Size passed and the ImageSizeRatioX member (328) of the Item passed and whose height is the product of the height of the Size passed and the ImageSizeRatioY member (329) of the Item passed (this is the Size at which the Item will be drawn) and then calls GetVisualItem (414) passing the Size calculated to obtain a VisualItem (362)

if the bBitmapValid member (363) of the VisualItem has a value of true then

{

transfers the pBitmap member (365) of the VisualItem directly to the Bitmap passed at the position passed by calling the BlitTo method (132) of the Bitmap passed and exits;

}

if the ItemArrayIndex member (326) of the Item passed is not NONE then

{

indexes an ItemArray (311) from the ArrayOfItemArrays (307) using the ItemArrayIndex member (326) of the Item passed and calls the DrawItemArray

process (410) passing the ItemArray indexed, the pBitmap member (365) of the VisualItem and the Size value calculated;

}

otherwise if the ImageIndex member (325) of the Item passed is not NONE then

{

indexes a Bitmap from the ReferenceImageArray (306) using the ImageIndex member (325) of the Item passed and stretches the indexed Bitmap onto the pBitmap member (365) of the VisualItem by calling its StretchToFit method (133);

}

otherwise

{

paints a blank pattern onto the pBitmap member (365) of the VisualItem by calling its FillColor method (131);

}

Finally DrawItem process draws a frame onto the pBitmap member (365) of the VisualItem using its member functions, sets the bBitmapValid member (363) of the VisualItem to a true value, transfers the pBitmap member (365) of the VisualItem directly to the Bitmap passed at the position passed by calling the BlitTo method (132) of the Bitmap passed and exits;

GetVisualItem.

The GetVisualItem process (414) is passed an Item (322) and a Size value and:

if the VisualItemIndex member (333) of the Item passed has a value of NONE then

{

creates a new VisualItem (362) and adds it to the VisualItemsArray (345) setting the VisualItemIndex member (333) of the Item passed equal to the index of the new VisualItem in the VisualItemsArray;

}

otherwise

{

indexes an VisualItem (362) from the VisualItemsArray (345) using the VisualItemIndex member (333) of the Item passed and

if the BitmapSize member (364) of the VisualItem indexed is equal to the Size passed then

exits returning the VisualItem indexed;

otherwise

deletes the pMemBitmap member (365) of the VisualItem indexed;

}

The GetVisualItem process then creates a new Bitmap (126) for the pMemBitmap member (365) of the VisualItem created or indexed by calling the Bitmap(Size) constructor (130) passing the Size value passed, sets the BitmapSize member (364) of the VisualItem equal to the Size passed and the bBitmapValid member (363) to false and exits returning the VisualItem created or indexed.

Invalidate

The Invalidate process (415) is passed an Item (322) and an ItemArray (311) and is called when for any reason when due to some aspect of dynamic response an Item or ItemArray has changed its appearance.

One of the passed parameters is always NULL and the Invalidate process operate on the other non NULL parameter, this allows it to be called with either an Item or an ItemArray as the operand. The purpose of the process is to set the bBitmapValid member (363) of the VisualItem (362) holding an image of the operand and all VisualItems holding images of its ancestors to a false value, forcing them to be redrawn by the DrawItem process (413) the next time that the display is redrawn.

The Invalidate process operates as follows:

if the Item passed is not NULL then

{

if the VisualItemIndex member (333) of the Item passed is not NONE then

indexes an VisualItem (362) from the VisualItemsArray (345) using the VisualItemIndex member (333) of the Item passed, and sets the bBitmapValid member (363) of the VisualItem indexed to false;

(N.B. if there is no VisualItem then by being non-existent it is implicitly invalid);

}

the process then calls the GetParentItem process (427) passing the parameters passed to itself and if the return from GetParentItem is not NULL then

makes a recursive call to itself passing the Item returned by the call to GetParentItem

The recursive calls by the Invalidate process ensure that all ancestors of the Item or ItemArray invalidated are also invalidated and terminate when the call to GetParentItem returns a NULL.

GetZElement

The GetZElement process is passed a ZArray and an Item and returns the ZElement associated with the Item passed. It does this by iterating through all the ZElements in the ZArray passed and for each ZElement:

if the ItemIndex member of the ZElement s equal to the OwnIndex member of the Item passed then

exits returning that ZArray

Control unit processes

HandleEvents

The HandleEvents (416) process is called by the SendEvent process (127) of the host application (404) it is the only call made by the host application to the Core Engine (405) and is passed a message code describing the event, a set of parameters (described below) and a Bitmap representing the display on which the Core Engine can draw any changes that result from the event handled. When redrawing the display, the process uses a memory Bitmap referenced by g_pWindowMemBitmap (349) which is passed to the DrawItemArray (410) process, this Bitmap is then transferred to the display device in a single BitBlit operation, thus avoiding the flicker cause by sequences of drawing operation being made directly to the screen.

Figure 27. is a flow chart showing steps in the operation of the HandleEvents process (416) details of each step according to the box in which they appear are as follows:

530. The HandleEvents process is called and is passed the following named parameters:

message which is an integer holding a message or event code,
x and y which are integers used to pass position and size information,
pVoid1 which is a pointer used to reference message specific information,
a Bitmap
and pVoid2 which is a pointer used to reference further message specific information,

operation then continues from the box numbered (531).

531. represents the sequence of execution being determined by the value of the message parameter and following the direction indicated by the box showing the value that matches the message:

532. message has the value INITIALISE, continues from (536);

533. message has the value WM_DROPFILES, continues from (537);

534. message has the value WM_SIZE continues from (538);

535. message has none of the above values continues from (541);

536. In response to INITIALISE , Initialises all global variables setting g_pItemArrayShowing (346), g_pItemShowing (347) , g_pCapturedItem (350) and g_pCapturedItemArray (351) equal to NULL;
setting g_bCaptureOn (352) equal to false;
and setting g_ArrayIndexLoading (308), g_ItemIndexLoading (309), g_NumLoading (410) and g_ShowingNestDepth (348) equal to 0, the process then exits.

537. In response to DROPFILES, calls the Populate process (407) passing an ItemAddress (317) with both members set to NONE (indicating that there is no parent), a StringList de-referenced from the pVoid1 parameter passed and a NestDepth value of zero, calls the Arrange process (408), points g_pItemArrayShowing (346) at the zero element of ArrayOfItemArrays (307), and sets g_ShowingNestDepth (348) to zero, operation then continues from the box labelled 'Draw' and numbered (558)

538. In Response to WM_SIZE, if g_pWindowMemBitmap (349) does <u>not</u> have a null value

then

{

539. deletes g_pWindowMemBitmap (it is now the wrong size);

}

540. sets g_WindowSize (366) to have a width equal to the x parameter passed and a height equal to the y parameter passed.

(N.B a new Window Bitmap of the correct size will be created in step at box 556)

operation then continues from the box labelled 'Draw' and numbered (558)

541. if g_pItemArrayShowing (346) does not have a null value then

operation then continues from the box numbered (542)

otherwise

the process exits.

542. represents the sequence of execution being determined by the value of the message parameter and following the direction indicated by the box showing the value that matches the message:

543. message has the value BACKBUTTON, continues from (550);

544. message has the value GOTFILEIMAGE, continues from (551);

546.message has the value WM_PAINT, continues from the box labelled 'Draw' and numbered (558)

547. message has any of the following values,

WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_RBUTTONDOWN, WM_RBUTTONUP, WM_LBUTTONDDBLCLK, WM_RBUTTONDDBLCLK LBUTTONDDRAG, RBUTTONDDRAG; continues from (552);

548. message has the value IDLE, continues from (553);

549. message has none of the above values, exits ;

550. In response to BACKBUTTON calls the Back process (424) and then continues from the box labelled 'Draw' and numbered (558)

551. In response to GOTFILEIMAGE calls the OnGotFileImage process (425) passing an Item de-referenced from the pVoid1 parameter passed, a Size value with its width set to the x parameter passed and its height set to the y parameter passed and a Bitmap de-referenced from the pVoid2 parameter passed, the process then continues from the box labelled 'Draw' and numbered (558)

552. In response to any mouse message creates a local boolean variable named bRedraw, calls the ItemArrayGetMouse process (417) passing g_pItemArrayShowing (346), g_WindowSize (366), a Point value representing the x and y parameters passed and the message passed, setting bRedraw equal to the value returned from ItemArrayGetMouse and;

553. if bRedraw has a true value;

continues from the box labelled 'Draw' and numbered (558)

otherwise

the process exits.

553. In response to IDLE calls the GetNextFileImage process (426) and exits.

554. In response to GOTFILEIMAGE calls the OnGotFileImage process (425) and continues from the box labelled 'Draw' and numbered (558)

558. The box labelled 'Draw', operation continues from the box numbered (542)

555. yes, if g_pWindowMemBitmap (349) does not have a null value;

556. create a new Bitmap (126) using the Bitmap(Size) constructor (130) and pass it , a Size value whose width is set to the x parameter passed and whose height is set to y parameter passed, set g_pWindowMemBitmap (349) to point at the new Bitmap;

557. call the FillColor method (131) of the Bitmap that pWindowMemBitmap points at to paint a background, call the DrawItemArray process (410) passing g_pItemArrayShowing (346), g_pWindowMemBitmap and g_WindowSize, call the BlitTo method (132) of the Bitmap that pMemBitmap points at, passing a Point value with x and y co-ordinates set to zero and g_pWindowMemBitmap as parameters.

ItemArrayGetMouse

The ItemArrayGetMouse process (417) handles all mouse events occurring over the display of a three dimensional virtual space representing an ItemArray (311). The role of the ItemArrayGetMouse process is to determine if the mouse is positioned over the image of an Item (322) within the display of the three dimensional virtual space, in which case it calls the ItemGetMouse process (422) , otherwise it calls the ItemArrayHandleMouse process (420).

The ItemArrayGetMouse process is called and is passed the following parameters:

an ItemArray (311) which is to receive the mouse event,
a Size value indicating the displayed size of the ItemArray;
a Point value indicating the position of the mouse within the display area of the ItemArray, or the amount the mouse has been dragged in the case of LBUTTONDDRAG and RBUTTONDDRAG events;
a message which is an integer holding a message or event code;

The process operates by obtaining a ZArray (353) from the return value of a call to the GetZArray process (411) passing the ItemArray passed, it then iterates through that

ZArray backwards (the opposite direction to that in the DrawItemArray process (410)) and for each ZElement (358) within it:

{

calls the HitTest process (418) passing the ItemArray and point passed and the ZElement and

if the HitTest process returns true then

{

calculates a position based on NodePosition member (360) of the ZElement and a default display size for Items displayed based on the Size passed in the same manner as the DrawItemArray process (410),

indexes an Item from the ItemArray passed using the ItemIndex member (359) of the ZElement and calls the ItemGetMouse process (422) passing the Item indexed, the default display size calculated, the position calculated and the message passed and then exits returning the value returned from the ItemGetMouse process;

}

}

if the HitTest process never returned a true throughout the iteration of the ZArray then the process calls the ItemArrayHandleMouse process (420) passing the same parameters passed to itself and then exits returning the value returned from the ItemArrayHandleMouse process.

However the simple iteration described above is modified to handle any Item which is 'On top' (that is it has been caused by a mouse click to be displayed in front of all others regardless of its position in the three dimensional virtual space) and is also modified to operate mouse capture.

The following operations are carried out before carrying out the general iteration of the ZArray described above and modify it as described below:

if g_bCaptureOn (352) has a value of true then

{

if g_pCapturedItemArray (351) points at the ItemArray passed then

{

calls the ItemArrayHandleMouse process (420) passing the same parameters passed to itself and then exits returning the value returned from the ItemArrayHandleMouse process;

}

otherwise

{

indexes an Item from the ItemArray passed using the ItemIndex member (359) of the ZElement which the pOnTopElement (357) of the ZArray points at and calls the ItemGetMouse process (422) passing the Item indexed, the default display size calculated, the position calculated and the message passed and then exits returning the value returned from the ItemGetMouse process;

(N.B. if it is not the ItemArray passed which has the capture then it must be the 'OnTop' Item since the mouse down event which initiates capture also causes the 'OnTop' effect)

}

}

otherwise

{

if the pOnTopElement (357) of the ZArray is not NULL

{

performs one pass of the iteration described above using the ZElement that the pOnTopElement of the ZArray points at;

}

the process then carries out the general iteration of the ZArray described above.

}

HitTest

The HitTest process (418) determines if the mouse is over the image of a Item associated with a single ZElement (358) within the display of a three dimensional virtual space representing an ItemArray (311). It is passed the following parameters:

an ItemArray for which the call is being made;
a point value indicating the position of the mouse within the display area representing the ItemArray passed;
a Size value indicating the displayed size of the ItemArray passed;
the ZElement for which a hit is being tested;

and operates as follows:

calculates a position based on NodePosition member (360) of the ZElement and a default display size for Items displayed based on the Size passed in the same manner as the DrawItemArray process (410), indexes an Item (322) from the ItemArray passed using the ItemIndex member (359) of the ZElement passed, calculates a Size value whose width is the product of the width of the Size passed and the ImageSizeRatioX member (328) of the Item indexed and whose height is the product of the height of the Size passed and the ImageSizeRatioY member (329) of the Item indexed in the same manner as the DrawItemProcess (413) and calculates a rectangle whose top left corner is the position calculated and whose width and height are equal to those of the Size value calculated,

if the point passed is within the rectangle calculated then

exits returning true;

otherwise

exits returning false.

GetZElement

The GetZElement process (419) returns the ZElement (358) associated with a given Item (322) within a given ZArray (353).

It is passed a ZArray in which the ZElement is sought; and an Item whose associated ZElement is sought. It iterates through the ZArray and for each ZElement:

{

if the ItemIndex member (359) of the ZElement is equal to the OwnIndex member (331) of the Item passed then

return that ZElement

}

if no match was found during the iteration then it returns NULL.

ItemArrayHandleMouse

The ItemArrayHandleMouse process (420) is called by the ItemArrayGetMouse process (417) when the mouse is not positioned over any Item within the ItemArray for which it has been called, that is it handles all mouse events occurring over unoccupied virtual three dimensional space. Its actions are to spin or tilt the virtual three dimensional space in response to a **left** button drag, scroll or translate the display in response to a **right** button drag and in response to a left button double click makes the ItemArray being handled into the root folder displayed.

Figure 28 is a flow chart showing steps in the operation of the ItemArrayHandleMouse process (420) details of each step according to the box in which they appear are as follows:

595. The ItemArrayHandleMouse process is called and is passed the following parameters:

an ItemArray (311) to which is to handle the mouse event,

a Size value indicating the displayed size of the ItemArray;

a Point value indicating the position of the mouse within the display area of the ItemArray, or the amount the mouse has been dragged in the case of LBUTTONDDRAG and RBUTTONDDRAG events;

a message which is an integer holding a message or event code;

596. declares bRedraw as a boolean local variable, sets it to false and continues operation from the box numbered (597)

597. if message has the value WM_LBUTTONDOWN or WM_RBUTTONDOWN then

598. sets g_bCaptureOn (352) to a 'true' value, set g_pCapturedItemArray (351) to point at the ItemArray passed, sets g_pCapturedItem (350) to a null pointer value, sets bRedraw to a 'true' value and continues operation from the box numbered (606);

otherwise

599. if message has the value LBUTTONDRAG and g_bCaptureOn (352) has a 'true' value and g_pCapturedItemArray (351) points at the ItemArray passed then

600. calls the SpinAndTilt process (421) passing the ItemArray, Size and point passed, sets bRedraw to a 'true' value and continues operation from the box numbered (606).

otherwise

601. if message has the value RBUTTONDRAG and g_bCaptureOn (352) has a 'true' value and g_pCapturedItemArray (351) points at the ItemArray passed then

602. increases the x and values of the ScrollOffset member (315) of the ItemArray passed, by the x and y values of the point passed scaled in the same manner as in the DrawItemArray (410), ItemArrayGetMouse (417) and HitTest (418) processes, calls the Invalidate process (415) passing a null pointer value and ItemArray passed, sets bRedraw to a 'true' value and continues operation from the box numbered (606);

otherwise

603. if message has the value LBUTTONDBLCLK then

604. set g_pItemArrayShowing (346) to point at the ItemArray passed, sets g_ShowingNestDepth (348) equal to the NestDepth member (321) of the ItemArray passed, sets bRedraw to a 'true' value and continues operation from the box numbered (605);

otherwise

605. sets g_bCaptureOn (352) to a 'false' value, sets g_pCapturedItemArray (351) to a null pointer value, sets g_pCapturedItem (350) to a null pointer value and continues operation from the box numbered (606).

606. exits the ItemArrayHandleMouse process returning bRedraw;

SpinAndTilt

The SpinAndTilt process (421) is called by the ItemArrayHandleMouse process (420) and if the horizontal movement of the mouse if greater than the vertical movement of the mouse then

rotates the three dimensional virtual space displayed with respect to the user

and otherwise

tilts the axis of rotation of the three dimensional virtual space towards or away from the user

It is passed the following parameters:

the ItemArray (311) to which is to be rotated;
a Size value indicating the displayed size of the ItemArray;
a Point value indicating amount the mouse has been dragged;

and operates as follows:

indexes a ZArray (353) from the ZArraysArray (344) using the ZArrayIndex member (320) of the ItemArray passed (there will be a ZArray indexed because an ItemArray cannot receive messages until it has been drawn at least once), sets the bOrientationValid member (355) of the ZArray indexed to false and if the pOnTopElement member (357) of the ZArray indexed, is not NULL then

{

indexes an Item (322) from the ArrayOfItemArrays (307) using the ItemIndex member (359) of the ZElement that the pOnTopElement member (357) of the ZArray indexed points at and calls the Select process (423) passing the Item indexed and a false value (this cancels any 'OnTop' effect);

}

the process then branches depending on whether the horizontal movement of the mouse is greater than whether the vertical movement of the mouse as follows:

if the square of the x value of the point passed is greater than the square of the y value of the point passed then

{

calculates a radian value R based on the x value of the point passed and scaled with respect to the width value of the Size passed and the mouse sensitivity, sets a value CosR equal to the cosine of R, sets a SinR value equal to the sine of R and, sets a matrix RY to equal a three by three matrix with the following values:

CosR	0	-SinR
0	1	0
SinR	0	CosR

premultiplies the SpinMatrix member (313) of the ItemArray passed, by RY;

}

otherwise

{

calculates a radian value R based on the y value of the point passed and scaled with respect to the height value of the Size passed and the mouse sensitivity, sets a value CosR equal to the cosine of R, sets a SinR value equal to the sine of R and, sets a matrix RY to equal a three by three matrix with the following values:

1	0	0
0	CosR	-SinR
0	SinR	CosR

premultiplies the Tilt Matrix member (314) of the ItemArray passed, by RY;

}

the process then call the Invalidate process passing a NULL Item value and the ItemArray passed and exits.

ItemGetMouse

The ItemGetMouse process (422) is called by the ItemArrayGetMouse process (417) when the mouse is positioned over an Item (322) within the ItemArray (311) for which it has been called, and handles events destined for that Item. Its actions are to call the

Select process (423) in response to a mouse down event, resize the Item in response to dragging its right or bottom edge, present a full view of the Item in response to a left button double click and if it is a folder then calls the ItemArrayGetMouse process (417) for the ItemArray representing that folder. Like the ItemArrayGetMouse process the ItemGetMouse process is complicated by its role in the operation of the capture mechanism.

Figure 29 is a flow chart showing steps in the operation of the ItemGetMouse process (422), details of each step according to the box in which they appear are as follows:

618. The ItemGetMouse process is called and is passed the following parameters:

an Item which is to receive the mouse event,
a Size value which is the default Size for Items within the ItemArray to which the Item belongs;
a Point value indicating the position of the mouse within the display area of the Item, or the amount the mouse has been dragged in the case of LBUTTONDDRAG and RBUTTONDDRAG events;
a message which is an integer holding a message or event code;

and operates as follows:

619. calculates a Size value whose width is the product of the width of the Size passed and the ImageSizeRatioX member (328) of the Item passed and whose height is the product of the height of the Size passed and the ImageSizeRatioY member (329) of the Item passed (this is the Size at which the Item is drawn by the DrawItem process (413)) sets a local boolean variable bRedraw to a 'false' value and continues operation from the box numbered (620);

620. if the message passed has the value WM_LBUTTONDOWN or WM_RBUTTONDOWN then

621. calls the Select process (423) passing the Item passed and a 'true' value, sets bRedraw to a 'true' value and continues operation from the box numbered (622);

otherwise

continues operation from the box numbered (624);

622. if the x value of the point passed is greater than the width value of the Size value calculated minus BORDERWIDTH, or the y value of point is greater than the height value of the Size value calculated minus BORDERWIDTH (where BORDERWIDTH has a value large enough for the user to have a reasonable margin of error when the locating the edges of a displayed Item) then

632. sets g_bCaptureOn (352) to a 'true' value, sets g_pCapturedItem (350) to point at the Item passed, sets g_pCapturedItemArray (351) to a NULL, sets bRedraw to a 'true' value and continues operation from the box numbered (634)

otherwise

continues operation from the box numbered (625)

624. if g_bCaptureOn (352) has a 'true' value and g_pCapturedItem (350) points at the Item passed then

continues operation from the box numbered (631)

otherwise

continues operation from the box numbered (625)

625. if the ItemArrayIndex member (326) of the Item passed has a value which is not NONE then

626. indexes an ItemArray (311) from the ArrayOfItemArrays (307) using the ItemArrayIndex member (326) of the Item passed, calls the ItemArrayGetMouse process (417) passing the ItemArray indexed, the calculated Size, and the point and message values passed, sets bRedraw equal to the return value from ItemArrayGetMouse and continues operation from the box numbered (634);

otherwise

continues operation from the box numbered (628)

628. if the message passed has the value WM_LBUTTONDBLCLK then

629. sets g_pItemShowing (347) to point at the Item passed, calls the ShowFile process (148) passing the Item passed and its Filename member (323) and continues operation from the box numbered (630)

630. sets g_bCaptureOn (352) to a 'false' value, sets g_pCapturedItem (350) equal to NULL, sets g_pCapturedItemArray (351) to NULL and continues operation from the box numbered (634)

631. if the message passed has the value LBUTTONDRAG then

632. sets the ImageSizeRatioX member (328) of the Item passed equal to the sum of the width value of the calculated Size value and the x value of the point passed, divided by the width value of the Size value passed, sets the ImageSizeRatioY member (329) of the Item passed equal to the sum of the height value of the calculated Size and the y value of point passed, divided

by the height value of Size value passed, and continues operation from the box numbered (633);

otherwise

continues operation from the box numbered (630)

633. calls the Invalidate process (415) passing the Item passed and a null pointer value and continues operation from the box numbered (634);

634. exits the ItemGetMouse process returning bRedraw;

Select

The Select process (423) operates the 'OnTop' effect which allows any one Item (322) within an ItemArray (311) to have the status of appearing on front of all others regardless of its Z position in the three dimensional virtual space in which it is displayed. The recursive action of the ItemGetMouse process (422) which calls the Select process operates the requirement that if an Item is made 'OnTop' within its parent ItemArray then all of its ancestor Items must be made 'OnTop' within their parent ItemArrays, however the Select process must handle the de-selection of the descendants of a folder Item which is deselected.

The Select process is passed the Item which is to be selected and a boolean value indicating 'true' if the Item is to be selected or 'false' if it is to be deselected;

and operates as follows:

Indexes the parent ItemArray of the Item passed from the ArrayOfItemArrays (307) using the ParentArrayIndex member (330) of the Item passed, obtains a ZArray (353) from a call to the GetZArray process (411) passing the parent ItemArray indexed and

if the pOnTopElement (357) of the ZArray obtained has a value of NULL (there is no 'OnTop' Item) then

{

if the boolean value passed is true then

{

obtains a ZElement (358) from a call to the GetZElement process (419) passing the ZArray obtained and the Item passed, sets the pOnTopElement (357) of the ZArray obtained to point at the ZElement obtained (this makes the Item passed 'OnTop'), calls the Invalidate process (415) passing the parent ItemArray indexed and exits;

(N.B. ancestors of the Item passed will have been set on top by making their own calls to the Select process as the mouse down event is propagated through them before being handled by the Item passed.)

}

otherwise

exits (no need to cancel an 'OnTop' state which doesn't exist);

}

if it has not exited the process it continues as follows:

if the boolean value passed is true then

{

if the ItemIndex member (359) of the ZElement that the pOnTopElement (357) of the ZArray obtained points at is equal the OwnIndex member (331) of the Item passed then

exits (it is already on top);

otherwise (it is not already on top)

{

obtains the 'OnTop' Item by indexing from the parent ItemArray indexed using the ItemIndex member (359) of the ZElement that the pOnTopElement (357) of the ZArray obtained points at, makes a recursive call to the Select process passing the 'OnTop' Item indexed and a boolean value of false (this deselects the Item currently on top),

obtains a ZElement (358) from a call to the GetZElement process (419) passing the ZArray obtained and the Item passed, sets the pOnTopElement (357) of the ZArray obtained to point at the ZElement obtained (this makes the Item passed 'OnTop'), calls the Invalidate process (415) passing the Item passed and exits;

}

}

otherwise (the boolean value passed is false)

{

calls the Invalidate process (415) passing the parent ItemArray indexed and if the ItemIndex member (359) of the ZElement that the pOnTopElement (357) of the ZArray obtained points at is <u>not</u> equal the OwnIndex member (331) of the Item passed then

{

sets the pOnTopElement (357) of the ZArray obtained to NULL and exits (it was not the one on top);

}

otherwise (it is the one on top)

{

if the ItemArrayIndex (326) of the Item passed is NONE then

sets the pOnTopElement (357) of the ZArray obtained to NULL and exits (the on top Item was a file and has no descendants);

}

}

if it has not exited (the boolean value passed is false, it is the one on top and it is a folder) the process continues as follows:

sets the pOnTopElement (357) of the ZArray obtained to NULL, indexes the child ItemArray of the Item passed from the ArrayOfItemArrays (307) using the ItemArrayIndex member (326) of the Item passed, obtains a child ZArray (353) from a call to the GetZArray process (411) passing the child ItemArray indexed and

if the pOnTopElement (357) of the child ZArray obtained is NULL

exits (the folder represented by the Item passed does not have any on top elements)

otherwise (the folder the Item passed represents has an on top element)

{

indexes the 'OnTop' child Item from the child ItemArray indexed using the ItemIndex member (359) of pOnTopElement (357) of the child ZArray obtained, makes a recursive call to the Select process passing the 'OnTop' child Item indexed and a boolean value of false and exits;

(N.B. recursive propagation of this call to Select operates the de-selection of the descendants of a folder Item which is deselected)

}

Back

The Back process (424) reverses the effect of double clicking on a displayed folder or file.

Whatever file or folder is currently filling the display area, the Back process causes its parent folder to fill the display area instead, except in the case that the root folder is currently filling the display area in which case it does nothing.

The Back process operates as follows:

If g_pItemShowing (347) is not NULL (there is a view of a file filling the display area) then

{

calls the CloseFileShowing process (149), points g_pItemArrayShowing (346) at an ItemArary (311) indexed from the ArrayOfItemArrays (307) using the ParentArrayIndex member (330) of the Item that pItemShowing (347) points at, sets g_pItemShowing to NULL and exits;

}

otherwise if the ParentArrayIndex member (318) of the ParentItemAddress member (316) of the ItemArray (311) that g_pItemArrayShowing (346) points at is NONE (there is no parent - it is the root folder) then

exits;

otherwise

{

sets g_pItemArrayShowing (346) to point at the ItemArray (311) indexed from the ArrayOfItemArrays (307) using the ParentArrayIndex member (318) of the ParentItemAddress member (316) of the ItemArray that g_pItemArrayShowing currently points at, sets g_ShowingNestDepth (348) equal to the NestDepth member (321) of the new ItemArray (311) that g_pItemArrayShowing (346) now points at and exits.

}

The following describes modifications which are necessary to effect the enhancement of the application so as to display links as illustrated in figure 19.

WebBrowser control modifications

The WebBrowser control (128) is modified so that when it handles the OnDocumentComplete event (151), it calls a new GetLinks process (170) which

constructs a StringList containing the location of all files that the page that it holds is linked to and calls the SendEvent process (127) passing the following parameters:

message = DOITEMLINKS, x = 0, y = 0,
pVoid1 = the PointerValue member (150) of the WebBrowser control (128),
pVoid2 = pointing at the StringList constructed by the new GetLinks process (170);

this is done before the WebBrowser control posts a message to destroy itself.

HandleEvent modifications

The HandleEvent process (416) is modified by inserting a test for a message value of DOITEMLINKS before the existing test for a message value of GOTFILEIMAGE and if the message has a value of DOITEMLINKS then calls the new DoItemLinks process (30) passing an Item (322) de-referenced from the pVoid1 parameter passed and a StringList de-referenced from the pVoid2 parameter passed.

DrawItemArray modifications

The DrawItemArray process (410) is modified by inserting a call to the new DrawArrayLinks process (32) passing the ItemArray, Bitmap and Size value passed, before exiting.

GetZArray modifications

The GetZArray process (411) is modified by inserting a call to the new SortItemZIndices process (35) passing the ZArray obtained or created, after making the existing call to the ZSort process (412).

Modifications to the ItemArrayHandleMouse (420) and ItemGetMouse (422) processes will be discussed after discussing the extensions to the data structures (40), as the

modifications to those processes make reference to members which extend the data structures.

Data structure modifications

Figure 30 is a block diagram showing the additions to the data structures to effect the enhancement.

The Item structure (322) is extended by addition of the following:

NearLinks (370) which is an array of NearLinkElements (375) each of which indicates a hyperlink to another Item residing in the same folder, each NearLinkElement consisting of:

a ToIndex member (376) which is an index into the common parent ItemArray (311) of the Item to which it links,

and a SelectOn member (377) which is a boolean holding a value of 'true' if the link is one of the links from the Item selected for the purpose of highlighting its links, otherwise it holds 'false'.

FarLinks (371) which is an array of FarLinkElements (378) each of which indicates a hyperlink to another Item not residing in the same folder, each FarLinkElement consisting of:

a ToArrayIndex member (380) which is an index into the ArrayOfItemArrays (307) of the ItemArray in which the Item to which it links resides,

and a ToIndex member (379) which is an index into the ItemArray indexed by the ToArrayIndex member (380), of the Item to which it links;

BridgeLinks (372) which is an array of BridgeLinkElements (378) each of which indicates a displayed Bridge link which is not a hyperlink itself but is implied by hyperlinks within the system of files and folders being displayed, each BridgeLinkElement consisting of:

a ToIndex member (382) which is an index into the common parent ItemArray (311) of the Item to which it links,

and a SelectOn member (383) which is a boolean holding a value of 'true' if the link is a part of a path from the Item selected for the purpose of highlighting its links, otherwise it holds 'false'.

NumLinksOut (373) which is an integer recording the need for the display of an external link indicator - an external link indicator is displayed if NumLinksOut is greater than zero;

LinksOutSelectOn (374) which is a boolean holding a value of 'true' if the external link indicator is a part of a path from the Item selected for the purpose of highlighting its links, otherwise it holds 'false'.

The ItemArray structure (311) is extended by the addition of a pLinkSelectedItem member (389) which is a pointer that either holds a value of NULL, or points to the Item selected for the purpose of highlighting its links - the existence of a non-NULL value held by this member is a faster indication of whether the Item selected for the purpose of highlighting its links is within a given ItemArray than iterating that given ItemArray to see if any of its Items match the global variable g_ pLinkSelectedItem (385) described below.

The Visual Memory Globals (343) are is extended by addition the following:

g_ pLinkSelectedItem (385) which is a pointer that either holds a value of NULL, or points to the Item selected for the purpose of highlighting its links;

g_LinkSelectLocked (386) which is a boolean holding a value of 'true' if the user has chosen to hold an Item selected for the purpose of highlighting its links whilst performing other operations such as rotation of one or more virtual three dimensional spaces.

The ZArray structure (353) is extended by the addition of a ItemZElementIndices member (384) which is an array of integers arranged so that the index within the ZArray of the ZElement (358) associated with a given Item (322) is indexed from the new ItemZElementIndices member (384) using the OwnIndex member (331) of that given Item - the new ItemZElementIndices member (384) is provided to avoid the need for iteration in obtaining the ZElement (358) associated with a given Item (322).

Modifications to the ItemArrayHandleMouse (420) and ItemGetMouse (422) processes will now be described.

ItemArrayHandleMouse modifications

With reference to figure 28 the ItemArrayHandleMouse process (420) is modified by inserting the following operations after initialising the bRedraw local variable (596) and before the test for a mouse down event (597):

if g_pLinkSelectedItem (385) does not have a NULL value g_bLinkSelectLocked (386) has a false value then

{

calls the new HighlightItemLinks process (36) passing the Item that g_pLinkSelectedItem (385) points at, and a boolean value of false, calls the Invalidate process (415) passing a NULL pointer value and the ItemArray passed to the ItemArrayHandleMouse process and sets bRedraw to a true value;

(N.B. the call to the new HighlightItemLinks process (36) passing a false, cancels the Highlighted Item if there is one, unless it has been locked by the user - this being done because the mouse is no longer positioned over the highlight hotspot at the top left corner of a displayed Item.)

}

operation then continues from the box numbered (597).

ItemGetMouse modifications

With reference to figure 29 the ItemGetMouse process (422) is modified as follows:

the following operations are inserted after the call to the Select process (621) and before testing if the mouse is over the right or bottom edge (622):

if the x member of the point passed has a value less than BORDERWIDTH and the y member of the point passed has a value less than BORDERWIDTH (where BORDERWIDTH has a value large enough for the user to have a reasonable margin of error when the locating the top left corner a displayed Item) then:

{

if g_bLinkSelectLocked (386) has a true value then

{

sets g_bLinkSelectLocked to false;

}

otherwise (g_bLinkSelectLocked does not have a true value)

{

sets g_bLinkSelectLocked to true;

}

(N.B a mouse click over the top left corner of a displayed Item toggles whether the Item selected for the purpose of highlighting its links is held or locked in that status while other mouse operations are performed)

}

operation then continues from the box numbered (622).

the following operations are inserted after failing the test for the Item being a folder (625) and before testing for a double click of the mouse (628):

if the message passed is WM_MOUSEMOVE then

{

if the x member of the point passed has a value less than BORDERWIDTH and the y member of the point passed has a value less than BORDERWIDTH (where BORDERWIDTH has a value large enough for the user to have a reasonable margin of error when the locating the top left corner a displayed Item) then

{

if g_bLinkSelectLocked (386) has a false value and g_pLinkSelectedItem (385) does not point at the Item passed then

{

calls the HighlightItemLinks process (36) passing the Item passed and a boolean value of true, sets bRedraw to a true value and continues operation from the box numbered (628);

(N.B. the call to the new HighlightItemLinks process (36) passing a true sets the Item passed to be the Item selected for the purpose of highlighting its links)

}

otherwise

continues operation from the box numbered (628);

}

otherwise (if the point passed is not positioned over the top left corner of the displayed Item within a reasonable margin of error)

{

if g_pLinkSelectedItem (385) does not have a NULL value and g_bLinkSelectLocked (386) has a false value then

{

calls the HighlightItemLinks process (36) passing the Item (322) that g_pLinkSelectedItem (385) points at and a boolean value of false, sets bRedraw to a true value and continues operation from the box numbered (628);

(N.B. the call to the new HighlightItemLinks process (36) passing a false, cancels the Highlighted Item if there is one, unless it has been locked by the user - this being done because the mouse is no longer positioned over the highlight hotspot at the top left corner of a displayed Item.)

}

otherwise

continues operation from the box numbered (628);

}

}

otherwise (if the message passed is not WM_MOUSEMOVE)

continues operation from the box numbered (628);

The new processes of the second embodiment of the invention will now be described in more detail.

DoItemLinks

The DoItemLinks process (30) is passed an Item (322) and a StringList holding all the links from that Item, the process populates the NearLinks (370) and FarLinks (371) members of the Item passed according to the elements of the StringList passed and calls the DoFarLink process (31) for each FarLink (378) added to the FarLinks member of the Item passed. For each link the process caries out a search of every Item in the data model to determine if it is the Item at the other end of the link

The DoItemLinks process (30) operates as follows:

If the StringList passed is empty then the process exits immediately;
otherwise:

for each member of the StringList passed, iterates through every Item (322) in every ItemArray (311) in the ArrayOfItemArrays (307) and for each Item iterated:

{

if the filename member (323) of the current iteration Item is equal to that member of the StringList then

{

if the ParentArrayIndex member (330) of the current iteration Item is equal to the ParentArrayIndex of the Item passed (they are in the same folder) then

{

creates a new NearLinkElement (375), sets the ToIndex member (376) of the new NearLinkElement equal to the OwnIndex member (331) of the current iteration Item, sets the SelectOn member (377) of the new NearLinkElement to false, and adds the new NearLinkElement to the NearLinks member (370) of the Item passed;

}

otherwise (they are <u>not</u> in the same folder)

{

creates a new FarLinkElement (378), sets the ToIndex member (379) of the new FarLinkElement equal to the OwnIndex member (331) of the current iteration Item, sets the ToArrayIndex member (380) of the new FarLinkElement equal to the ParentArrayIndex (330) member of the current iteration Item, adds the new FarLinkElement to the FarLinks member (371) of the Item passed and calls the DoFarLink process (31) passing the Item passed and the current iteration Item;

}

and then in either case terminates further iteration of Items for that member of the StringList passed and starts the iteration of Items for the next member of the StringList passed;

}

otherwise (the filename member of the current iteration Item is not equal to that member of the StringList)

continues the iteration of Items for that member of the StringList;

}

DoFarLink

The DoFarLink process (31) is passed a 'From' Item (322) and a 'To' Item. The process adds BridgeLinks and external link indicators to Items throughout the data structure as required to indicate the full path of the link from the 'From' Item (322) to the 'To' Item. The requirements of its operation can be understood with reference to figure 31 in which a first Item (800) is linked to a second Item (801) which is not located in the same folder. To illustrate a general case both Items are deeply nested, but the second Item (801) is more deeply nested than the first Item (800). Folder Items 802, 803 and 805 to 808 show external link indicators, folder Items 804 and 809 are implicitly linked by a BridgeLink (810) and folder Item 811 is the most deeply nested common ancestor of both the first and second Items within which the entire path of the link is displayed.

The DoFarLink process iterates backwards through the ancestry of the most deeply nested Item until the nest depth of the other Item has been reached, it then iterates backwards through the ancestry of both Items until it reaches a common parent and then within that common parent adds a BridgeLink connecting its child which is an ancestor of the first Item with its child which is an ancestor of the second Item.

Operation of the DoFarLink process (31) proceeds as follows:

A local pointer variable pItemA is created and set to point at the 'From' Item passed and a local pointer variable pItemB is created and set to point at the 'To' Item passed;

while the NestDepth member (332) of the Item that pItemA points at is not equal to the NestDepth member (332) of the Item that pItemB points at it does the following:

{

if the NestDepth member of the Item that pItemA points at is greater than the NestDepth member of the Item that pItemB points at then

{

increments the value of the NumLinksOut member (373) of the Item that pItemA points at by one, calls the GetParentItem process (427) passing the Item that pItemA points at and sets pItemA to point at the Item returned by the call to the GetParentItem process;

}

otherwise

{

increments the value of the NumLinksOut member (373) of the Item that pItemB points at by one, calls the GetParentItem process (427) passing the Item that pItemB points at and sets pItemB to point at the Item returned by the call to the GetParentItem process;

}

(N.B. here it iterates backwards through the ancestry of the most deeply embedded Item)

}

when the while loop above has completed and the NestDepth members of the Items that pItemA and pItemB point at have the same value the process continues as follows:

while the ParentArrayIndex member (330) of the Item that pItemA points at is not equal to the ParentArrayIndex member (330) of the Item that pItemB points at it does the following:

{

increments the value of the NumLinksOut member (373) of the Item that pItemA points at by one, calls the GetParentItem process (427) passing the Item that pItemA points at sets pItemA to point at the Item returned by the call to the GetParentItem process, increments the value of the NumLinksOut member (373) of the Item that pItemB points at by one, calls the GetParentItem process (427) passing the Item that pItemB points at and sets pItemB to point at the Item returned by the call to the GetParentItem process;

(N.B. here it iterates backwards through the ancestry of both Items simultaneously until they have a common parent)

}

when the while loop above has completed and the ParentArrayIndex members (330) of the Items that pItemA and pItemB point at have the same value the process continues as follows:

creates a new BridgeLinkElement (381), sets the ToIndex member (382) of the new BridgeLinkElement equal to the OwnIndex member (331) of the Item that pItemB points at, sets the SelectOn member (383) of the new BridgeLinkElement to false, adds the new BridgeLinkElement to the BridgeLinks member (372) of the Item that pItemA points at and exits.

DrawArrayLinks

The DrawItemArrayLinks process (32) takes care of drawing all the links within a displayed ItemArray (311), it is passed an ItemArray, a Bitmap (126) on which to draw and a Size value.

The process operates as follows:

calls the GetZArray process (411) passing the ItemArray passed to obtain a ZArray (353) and then

iterates through every ZElement (358) within that ZArray and for each ZElement:

{

indexes an Item (322) from the ItemArray passed using the ItemIndex member (359) of the ZElement and calls the DrawItemLinks process (33) passing the ItemArray passed, the Item indexed, the ZArray obtained, the Bitmap passed and the Size passed;

}

if the pLinkSelectedItem member (389) of the ItemArray passed does not have a NULL value then

{

calls the DrawItemLinks process (33) once more, passing the ItemArray passed, the Item that the pLinkSelectedItem member (389) of the ItemArray passed points at, the ZArray obtained, the Bitmap passed and the Size passed;

(N.B. this last call to the DrawItemLinks process ensures that selected links will be drawn last and therefore will not be obscured by other links.)

}

the process then exits.

DrawItemLinks

The DrawItemLinks process (33) draws all links and external link indicators that are visually connected to a given Item. It is passed an ItemArray (311) in within whose display the links are drawn, an Item (322) being the given Item, a ZArray (353) being

the ZArray associated with the ItemArray passed, a Bitmap (126) on which to draw and a Size value being the displayed size of the ItemArray passed and operates as follows:

calls the GetItemPos process (34) passing the ItemArray, ZArray, Item and size passed and sets a local Point variable ItemPos equal to the Point returned from the GetItemPos process (this obtains the position of the Item passed within the display of the ItemArray passed and holds it in the ItemPos local point variable) and then:

for each NearLinkElement (375) within the NearLinks member (370) of the Item passed:

{

indexes an Item from the ItemArray passed using the ToIndex member (376) of the NearLinkElement, calls the GetItemPos process (34) passing the ItemArray passed, the ZArray passed, the Item indexed and the Size passed setting a local Point variable ToItemPos equal to the Point value returned from the GetItemPos process and

if the SelectOn member (377) of the NearLinkElement is true then

{

calls the SetPen member function (134) of the Bitmap passed to choose a pen suitable for highlighted links;

}

otherwise

{

calls the SetPen member function (134) of the Bitmap passed to choose a pen suitable for non highlighted links;

}

and in either case draws a line onto the Bitmap passed from ItemPos to ToItemPos;

}

for each BridgeLinkElement (381) within the BridgeLinks member (372) of the Item passed:

{

indexes an Item from the ItemArray passed using the ToIndex member (382) of the BridgeLinkElement, calls the GetItemPos process (34) passing the ItemArray passed, the ZArray passed, the Item indexed and the Size passed setting a local Point variable ToItemPos equal to the Point value returned from the GetItemPos process and

if the SelectOn member (383) of the BridgeLinkElement is true then

{

calls the SetPen member function (134) of the Bitmap passed to choose a pen suitable for highlighted links;

}

otherwise

{

calls the SetPen member function (134) of the Bitmap passed to choose a pen suitable for non highlighted links;

}

and in either case draws a line onto the Bitmap passed from ItemPos to ToItemPos;

}

if the NumLinksOut member (373) of the Item passed is greater than zero then

{

if the LinksOutSelectOn member (374) of the Item passed has a value of 'true' then

{

calls the SetPen member function (134) of the Bitmap passed to choose a pen suitable for highlighted links, draws a line from ItemPos to the top left corner of the display of the ItemArray passed and exits;

}

otherwise

{

calls the SetPen member function (134) of the Bitmap passed to choose a pen suitable for non highlighted links, draws an external link indicator attached to the Item passed using ToItemPos to locate its position and exits;

}

}

otherwise

exits

GetItemPos

The GetItemPos process (34) is passed an ItemArray (311), a ZArray (353), an Item (322), and a Size value, returns the screen position of the Item passed within the display of the ItemArray passed and operates as follows:

indexes an integer from the ItemZIndices member (384) of the ZArray passed using the OwnIndex member (331) of the Item passed, indexes a ZElement (358) from the ZArray passed using the integer indexed;

and then calculates a position based on NodePosition member (360) of the ZElement, scaled to fit the view area (indicated by the Size passed), offset to place the origin in the centre and further offset according to the value of the ScrollOffset member (315) of the ItemArray passed in the same manner as the DrawItemArray process (410);

and then exits returning the position calculated.

SortItemZIndices

The SortItemZIndices process (35) is passed a ZArray (353) and maintains the ItemZIndices member (384) of that ZArray so that indexing into it with the OwnIndex

member (331) of an Item will obtain the index within that ZArray of the ZElement (358) associated with that Item. The process operates as follows:

if the ItemZIndices member (384) of the ZArray passed is empty then

{

populates the ItemZIndices member (384) of the ZArray passed with one element for each ZElement in the ZArray (the initial values of the element of the ItemZIndices member (384) of the ZArray passed do not matter) ;

}

for each ZElement (358) in the ZArray passed:

{

sets the element of the ItemZIndices member (384) of the ZArray passed whose index into the ZArray passed is equal to the ItemIndex member (359) of the ZElement, equal to the index into the ZArray passed of the ZElement;

}

the process then exits;

HighlightItemLinks

The HighlightItemLinks process (36) is called to highlight the full path of all links from a given Item (322). It is passed an Item (322) and a boolean value and if the boolean value is false it removes all highlight from the full path of all links from that Item. It also records which Item is selected for this purpose using g_pLinkSelectedItem (385) and maintains correct values for the pLinkSelectedItem member (389) of the parent ItemArray (311) of the Item passed.

It operates as follows:

if the boolean value passed is 'true' then

{

if g_pLinkSelectedItem (385) does not have a NULL value then

{

it recursively calls the HighlightItemLinks process (36) passing the Item that g_pLinkSelectedItem points at and a boolean value of 'false';

(N.B. this deselects any Item that was previously selected for this purpose)

}

and either way sets g_pLinkSelectedItem to point at the Item passed;

}

otherwise (the boolean value passed is 'false')

{

sets g_pLinkSelectedItem (385) to a NULL value;

}

either way, the process then indexes an ItemArray (311) from the ArrayOfItemArrays (307) using the ParentArrayIndex member (330) of the Item passed and

if the boolean value passed is 'true' then

{

sets the pLinkSelectedItem member (389) of the ItemArray indexed to point at the Item passed;

}

otherwise

{

sets the pLinkSelectedItem member (389) of the ItemArray indexed to a NULL value;

}

either way, the process then continues as follows:

for each NearLinkElement (375) in the NearLinks member (370) of the Item passed:

{

sets the SelectOn member (377) of the NearLinkElement equal to the boolean value passed;

}

and then

for each FarLinkElement (378) in the FarLinks member (371) of the Item passed:

{

indexes an ItemArray (311) from the ArrayOfItemArrays (307) using the ToArrayIndex member (380) of the FarLinkElement, indexes an Item (322) from ItemArray indexed using the ToItemIndex member (379) of the FarLinkElement and calls the HighlightFarLink process (37) passing the Item passed, the Item indexed and the boolean value passed;

}

with all iterations complete the process exits.

HighlightFarLink

The HighlightFarLink process (37) operates in a very similar manner to the DoFarLink process (31) except that instead of creating BridgeLinks and external link indicators it marks them as highlighted.

It is passed a 'From' Item (322), a 'To' Item and a boolean value, if the boolean value passed is false then the highlight is removed from the BridgeLinks and external link indicators concerned.

The process operates as follows:

calls the GetParentItem process (427) passing the 'From' Item passed and calls the Invalidate process (415) passing the Item returned from the call to the GetParentItem process,

calls the GetParentItem process (427) passing the 'To' Item passed and calls the Invalidate process (415) passing the Item returned from the call to the GetParentItem process,

(N.B. these calls to the Invalidate process ensure that all displays of ItemArrays affected by the addition or removal of highlight will be redrawn.)

A local pointer variable pItemA is created and set to point at the 'From' Item passed and a local pointer variable pItemB is created and set to point at the 'To' Item passed;

while the NestDepth member (332) of the Item that pItemA points at is not equal to the NestDepth member (332) of the Item that pItemB points at it does the following:

{

if the NestDepth member of the Item that pItemA points at is greater than the NestDepth member of the Item that pItemB points at then

{

sets LinksOutSelectOn member (374) of the Item that pItemA points at to the boolean value passed, calls the GetParentItem process (427) passing the Item that pItemA points at and sets pItemA to point at the Item returned by the call to the GetParentItem process;

}

otherwise

{

sets LinksOutSelectOn member (374) of the Item that pItemB points at to the boolean value passed, calls the GetParentItem process (427) passing the Item that pItemB points at and sets pItemB to point at the Item returned by the call to the GetParentItem process;

}

(N.B. here it iterates backwards through the ancestry of the most deeply embedded Item)

}

when the while loop above has completed and the NestDepth members of the Items that pItemA and pItemB point at have the same value the process continues as follows:

while the ParentArrayIndex member (330) of the Item that pItemA points at is not equal to the ParentArrayIndex member (330) of the Item that pItemB points at it does the following:

{

sets LinksOutSelectOn member (374) of the Item that pItemA points at to the boolean value passed, calls the GetParentItem process (427) passing the Item that pItemA points at sets pItemA to point at the Item returned by the call to the GetParentItem process,

sets LinksOutSelectOn member (374) of the Item that pItemB points at to the boolean value passed, calls the GetParentItem process (427) passing the Item that pItemB points at and sets pItemB to point at the Item returned by the call to the GetParentItem process;

(N.B. here it iterates backwards through the ancestry of both Items simultaneously until they have a common parent)

}

when the while loop above has completed and the ParentArrayIndex members (330) of the Items that pItemA and pItemB point at have the same value the process continues as follows:

for each BridgeLinkElement (381) within the BridgeLinks member (372) of the Item that pItemA points at:

{

if the ToIndex member (382) of the BridgeLinkElement is equal to the OwnIndex member (331) of the Item that pItemB points at then

sets the SelectOn member (383) of the BridgeLinkElement to the boolean value passed and exits;

(N.B. this is the BridgeLink associated with the FarLink being highlighted.)

}

the process exits.

It will be appreciated that an application described above could equally well be realised by constructing electronic logic circuitry which directly embodies the logic of the software described above.

It will further be appreciated that variations from the above described embodiments may still fall within the scope of the invention.

CLAIMS:

1. A method of displaying digitally stored information, comprising:

creating a virtual 3D space which is rotatable in response to user interaction;

representing items of the stored information as labelled nodes within the virtual 3D space; and

displaying the virtual 3D space to the user.

2. A method as claimed in claim 1, wherein each item of information is represented as an image located at a corresponding node, and which moves with the node as the virtual 3D space is rotated.

3. A method as claimed in claim 2, wherein each image is displayed so that it always appears as a 2D image facing the user, so that the user receives the impression of a plurality of 2D images suspended in 3D space.

4. A method as claimed in claim 2 or 3, wherein a first image obscures parts of other images with which it overlaps when the node of the first image is closer to the user than the nodes of the other images in the virtual 3D space.

5. A method as claimed in any preceding claim, wherein the virtual space is rotatable about a single axis in response to user interaction.

6. A method as claimed in claim 5, wherein the axis of rotation is tiltable towards or away from the user in response to user interaction.

7. A method as claimed in any preceding claim, wherein the rotation of the virtual space is controllable by the user activating a pointing device over an unoccupied part of the virtual space and moving the pointing device.

8. A method as claimed in any preceding claim, wherein the user can interact with any image using a pointing device to cause an action specific to that image or the

information which that image represents.

9. A method as claimed in claim 8, wherein the interaction with an image using the pointing device causes that image to be temporarily enlarged or displayed in a separate window.

10. A method as claimed in any preceding claim, wherein an action of a pointing device over an image displayed within a virtual space causes that image to be temporarily displayed so that it is not obscured by any other image.

11. A method as claimed in any preceding claim, wherein links between items of information are displayed as lines linking the nodes associated with those items of information.

12. A method as claimed in any preceding claim, wherein the items of digitally stored information include files arranged in folders, the method further comprising creating a further virtual 3D space representing the contents of a folder and displaying the further virtual 3D space to the user as an image attached to the node corresponding to that folder.

13. A method as claimed in claim 12, wherein the user can interact with the further virtual 3D space in the same way as with the virtual 3D space, the further virtual 3D space being embedded within the virtual 3D space.

14. A method as claimed in claim 12 or 13, wherein the embedded further virtual 3D space is displayable to the user in place of the virtual 3D space on selection by the user of said further virtual 3D space.

15. A method as claimed in claim 12 or 13, further comprising displaying within a virtual 3D space a line between an image corresponding to a first file and an image corresponding to a second file if the first and second files are linked.

16. A method as claimed in claim any of claims 12 to 15, further comprising displaying within a virtual 3D space a line between an image representing a file and an image representing a folder if the file is linked to one or more files embedded within the hierarchy represented by the folder.

17. A method as claimed in any of claims 12 to 16, further comprising displaying within a virtual 3D space a line between an image representing a first folder and an image representing a second folder if one or more files embedded within the hierarchy of the first folder are linked to one or more files embedded within the hierarchy of the second folder.

18. A method as claimed in any of claims 12 to 17, further comprising displaying an external link indicator attached to an image representing a first file if the first file is linked to one or more files located in parts of the hierarchy not contained within the parent folder of the first file.

19. A method as claimed in any of claims 12 to 18, further comprising displaying an external link indicator attached to an image representing a folder if a file located within the hierarchy of that folder is linked to one or more files located in parts of the hierarchy not contained within the parent folder of that folder.

20. A method as claimed in any of claims 12 to 19, wherein all external link indicators or lines representing links from a file or folder throughout the display are highlighted if the image corresponding to that file or folder is selected by the user.

21. A method of displaying digital information stored in the form of hierarchically arranged folders and files, comprising:

creating a first virtual 3D space for a root folder, each file and folder located within the root folder being associated with a unique point in the virtual 3D space;

displaying the first virtual 3D space to the user, each file or folder being represented by an image located at the associated unique point in the virtual 3D space;

wherein the image representing a folder within the first virtual 3D space is a

display of a further virtual 3D space representing the contents of that folder.

22. A method as claimed in claim 21, wherein the first virtual 3D space and the further virtual 3D spaces are rotatable in response to instructions from the user.

23. A computer storage medium having stored thereon a program arranged to perform the method of any preceding claim.

24. Apparatus for displaying a plurality of discrete items of information, comprising:

a digital display device controllable by a control unit;

wherein the control unit is arranged to cause images representing the items of information to be displayed on the display device so as to cause an impression to the user of a plurality of 2D images suspended within a virtual 3D space;

and wherein the control unit is further arranged to manipulate the display of the images so that the virtual 3D space appears to rotate in response to instructions from the user.

25. Apparatus as claimed in claim 24, wherein each image occupies a unique position in the virtual 3D space.

26. Apparatus as claimed in claim 24 or 25, wherein the control unit is arranged so that each image is drawn on the display device adjacent to a position which represents a 2D projection of its position within the virtual 3D space such that, when a first image is closer to the user in the virtual 3D space than a second image, the first image obscures any part of the second image occupying the same region of the display device.

27. Apparatus as claimed in claim 26, wherein the control unit is arranged so that the appearance of rotation of the 3D space to the user is caused by incremental variations of the angle from which the 2D projection is taken, each incremental variation being followed by re-drawing the images.

28. Apparatus as claimed in claim 27, wherein the control unit is arranged to hold a

memory image in random access memory of each 2D image to be displayed within a virtual 3D space, each memory image being in a form which can be quickly transferred to the display device or to another memory location;

and wherein the control unit is arranged such that each memory image is formed when the 2D images are first displayed, and during subsequent re-display each memory image is directly transferred to the display of the virtual 3D space.

29. Apparatus as claimed in claim 28, wherein the control unit is arranged so that, when a 2D image within a virtual 3D space is required to change its appearance, the memory image corresponding to that image is deleted or marked invalid and the display re-drawn; and wherein

during the re-draw of the display the deleted or invalid memory image is re-formed to contain the new 2D image and validated.

30. Apparatus as claimed in claim 29, wherein the control unit is arranged so that an image within the virtual 3D space can itself comprise a further virtual 3D space;

and so that if the memory image of an image displayed within the further virtual 3D space is deleted or marked invalid then the image in memory of the image within the virtual 3D space is also deleted or marked invalid.

31. Apparatus as claimed in any of claims 24 to 30, wherein the items of information include files and folders, a folder being represented as a discrete virtual 3D space, and wherein the control unit is arranged so that wherever there exists a link from a first file to a second file not sharing a parent folder with said first file, an iteration which operates on two files or folders is implemented, operating initially on the first and second file, and during each iteration

if one file or folder is more deeply embedded in the hierarchy than the other then an indication of external links held against the more deeply embedded file or folder is set and the next iteration is carried out replacing the more deeply embedded file or folder with its parent;

if both files or folders are embedded in the hierarchy equally deeply, but do not share the same parent folder, then an indication of external links held against both files

or folders is set and the next iteration is carried out replacing both files or folders with their parents; and

if both files or folders share the same parent folder then an implied link is generated between them by adding a reference to the file or folder deriving from the second file to the list of implied links held by the file or folder deriving from the first file and the iteration terminates.

32. A method substantially as herein described with reference to the accompanying drawings.

INVESTOR IN PEOPLE

Application No: GB 0208727.8 **Examiner:** Iwan Thomas
Claims searched: 1 and 24 **Date of search:** 9 October 2002

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.T):

Int Cl (Ed.7):

Other: Online: WPI, EPODOC, PAJ

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X	WO 00/73888A1	(IBM) See line 6 page 2 - line 23 of page 3 and figs. 5-7	1-3,5-7, 24 & 25
X	US 4685070A	(FLINCHBAUGH) See abstract	1-6,8, 24 & 25
X	JP2001204922A	(HEIWA) See abstract and figs. 20&22-24	1-6,24-27

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.